MANUAL DE REFERENCIA BASIC PARA EL PROGRAMADOR



Y

SPECTRAVIDEO™

SVI™



MANUAL DE REFERENCIA BASIC PARA EL PROGRAMADOR

MSX

Y

SPECTRAVIDEO™

SVI™



Versión en castellano

Editado por INDESCOMP, S.A.
Pº de la Castellana, 179 - 28016 Madrid

*Todas las consultas relativas a este libro deberán ser dirigidas a INDESCOMP, S.A.

Traduce, compone e imprime: CONORG, S.A.

I.S.B.N.: 84-86176-11-5

Depósito Legal: M-37060-1984

CONTENIDO



VERSIONES DE BASIC

Se dispone de dos versiones de BASIC: cassette y diskette; y en ambas el **Spectravideo** presenta las siguientes características:

* **Repertorio ampliado**
  Puede mostrarse en pantalla un repertorio de 102 diferentes caracteres, que incluyen las letras, cifras y signos convencionales, y además otra serie de símbolos que habitualmente se emplean en aplicaciones científicas y matemáticas.

* **Facilidades para gráficos**
  Estando instalado el procesador de imágenes de video TMS9918/9929 puedes dibujar puntos, rectas, arcos e incluso figuras completas. Hay además 52 símbolos gráficos accesibles mediante la pulsación de una u otra tecla LEFT GRPH o RIGHT GRPH simultáneamente con una de las 26 teclas de letras, con lo que se elige respectivamente el símbolo gráfico colocado a la **izquierda** o a la **derecha** de esa tecla.

  También es posible usar hasta 32 **"sprites"** -que 'llevan' cada uno y en planos diferentes una figura cuya forma es fácilmente definida por el usuario.

* **Facilidades de sonidos**
  Con el generador programable de sonidos AY8910, pueden conseguirse notas musicales sueltas con diversos tonos y prepararse melodías completas.

* **Mandos para juegos**
  En el lenguaje BASIC del Spectravideo, existen instrucciones para operar adecuadamente con **"joysticks"** de bastón y de palanca, con gatillo para disparos; y además, 'tableros graficadores' para realización de dibujos. La utilización de estos accesorios hace los programas más interesantes y divertidos.

1.1 BASIC para cassette

La versión de BASIC para cassette viene incorporada dentro de tu ordenador SVI, ocupando 32Ks de **memoria únicamente de lectura**, por lo que puedes usarlo sin tener que emplear nada de la **memoria de escritura y lectura** que tengas en tu ordenador; y que por tanto, puede ser dedicada exclusivamente a almacenar tus programas y tus datos. La capacidad disponible de memoria te será indicada dentro del **saludo** que aparece en pantalla al poner en marcha el ordenador.



El único equipo **periférico** de **almacenamiento** de información que puede usarse con esta versión de BASIC, es una **lecto-grabadora de cassette** standard.

1.2 BASIC para diskette

Esta versión de BASIC se suministra junto con otros programas en el diskette del sistema; y debe ser 'implantado' en la memoria de escritura y lectura de tu ordenador antes de que pueda usarse. Esta simple operación inicial se denomina **instauración del BASIC**, y el programa que la efectúa (implantador), viene grabado de forma permanente e indeleble en la memoria únicamente de lectura de tu ordenador, exigiendo aproximadamente 8Ks.

Al igual que en el caso anterior, la capacidad disponible de memoria de escritura y lectura para usar en tus programas y datos, se indica en el saludo que aparece en pantalla al poner en marcha el ordenador.

Con la versión de BASIC para diskette, se dispone de todas las facilidades de la versión de BASIC para cassette, y además de las correspondientes a la entrada y salida de información procedente de o con destino a la unidad de discos.



PUESTA EN MARCHA

2.1 Para iniciar el funcionamiento de tu ordenador, sigue los pasos correspondientes a la versión BASIC que tengas, enunciados a continuación:

2.1.1 Puesta en marcha del BASIC para cassette

Conecta al ordenador el monitor de video o aparato TV, de acuerdo con las instrucciones dadas en el Manual de Usuario. Para **cargar** programas desde cinta o **guardar** un programa en la cinta, simplemente conecta tu lecto-grabadora de cassettes al ordenador, insertando el conector montado en el extremo del cable correspondiente al cassette dentro del enchufe incorporado en el lateral derecho del ordenador. Luego, simplemente acciona el interruptor de encendido, después de haber conectado adecuadamente la alimentación.

Si empleas la versión BASIC para cassette, teniendo conectados a tu ordenador unidades lecto-grabadoras de diskette, asegúrate que no hay colocado ningún diskette en la unidad de disco 1, o que si lo está has dejado abierta la puertecilla de la unidad 1.

Al encender, verás que el ordenador te "saluda" con el mensaje:

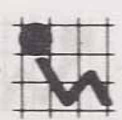
SV-extended BASIC version 1.0
Copyright 1983(c) by Microsoft Corp.
XXXXX Bytes free
OK
■

siendo XXXXX sustituido por el número correspondiente a la capacidad de memoria disponible para tus programas.

Asimismo, y en la parte inferior aparecen enumeradas las 'series de comandos' asociadas a las cinco primeras **teclas** funcionales programables por el usuario.

2.1.2 Puesta en marcha del BASIC para diskette

Conecta tu unidad de expansión, tu unidad de adaptación (si no está incorporado en la unidad de expansión) y las unidades de disco a tu ordenador. No olvides obviamente, conectar el aparato TV o monitor de video. Consulta el manual de usuario correspondiente a la unidad de expansión y a las unidades de disco para más detalles.



Enciende el aparato TV o el monitor de video, y la unidad de expansión. La ductora de diskette producirá ruidos y chasquidos, pero son habituales. Antes de encender el ordenador, inserta el diskette del sistema BASIC en la unidad de disco número 1, con la etiqueta del diskette mirando hacia arriba y hacia la 'boca' de la unidad de disco. Cierra la puertecilla frontal de la unidad de discos. Se encenderá un piloto rojo rotulado "IN USE"; y aparecerá en la pantalla el saludo del ordenador, con el mensaje:

SV-extended BASIC version 1.1
Copyright 1983(c) by Microsoft Corp.
Disk versión 1.0 by Microsoft Cop.
XXXXX Bytes free
OK
■

que se diferencia del anterior en que indica versión diskette.

Asimismo, en la parte inferior se enumeran las 'series de comandos' asociadas a las cinco teclas funcionales programables por el usuario.

Siempre que se produzca un fallo, vuelve a instaurar el sistëma, i.e.: apaga el ordenador y vuelve a encenderlo.

2.2 MODOS DE OPERACION

Una vez que hayas puesto en marcha el ordenador y el BASIC esté 'instaurado', te saludará con el mensaje mencionado. Durante las operaciones subsecuentes, te indicará mediante el mensaje

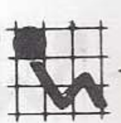
OK

que está **PREPARADO** para recibir tus comandos e instrucciones. Cuando se encuentra en esta situación, decimos que el ordenador está "a nivel de comando".

Puedes comunicar tus mandatos al BASIC en un modo de los dos posibles: directo o indirecto.

2.2.1 Modo Directo

En este modo, tus mandatos no van precedidos por un número de línea, y el BASIC los ejecuta **inmediatamente**. Se suele decir que es un **comando**.



En el modo directo puedes hacer que efectúe las siguientes tareas: cálculos aritméticos, operaciones lógicas, asignación de valores a variables (conservados para uso posterior), y prácticamente todas las frases admitidas en el lenguaje BASIC. Recuerda sin embargo, que los **comandos** se obedecen inmediatamente, pero no se conservan en memoria aunque aparezcan en pantalla. Por ejemplo:

```
LET A = 34 |ENTER|
OK
PRINT A |ENTER|
34
OK
```

Nota: Pulsar la tecla marcada 'ENTER' significa que has concluido tu comando y que el ordenador pasará inmediatamente a efectuar la acción que le has mandado.

2.2.2 Modo Indirecto

En el modo indirecto se precede cada una de las frases del BASIC de un número de línea identificativo, y queda almacenada en el lugar correspondiente de la memoria para constituir un programa. Se denominan usualmente **instrucciones.**

El programa así formado, solamente será ejecutado por el ordenador cuando le teclees el **comando** RUN. Por ejemplo:

```
10 LET A = 34 |ENTER|
20 PRINT A |ENTER|
RUN |ENTER|
34
OK
```

Nota: Pulsando la tecla 'ENTER' significa que has concluido una instrucción y que el ordenador sólo tiene que almacenarla en el lugar correcto de la memoria. Habitualmente decimos **ADENTRO.**

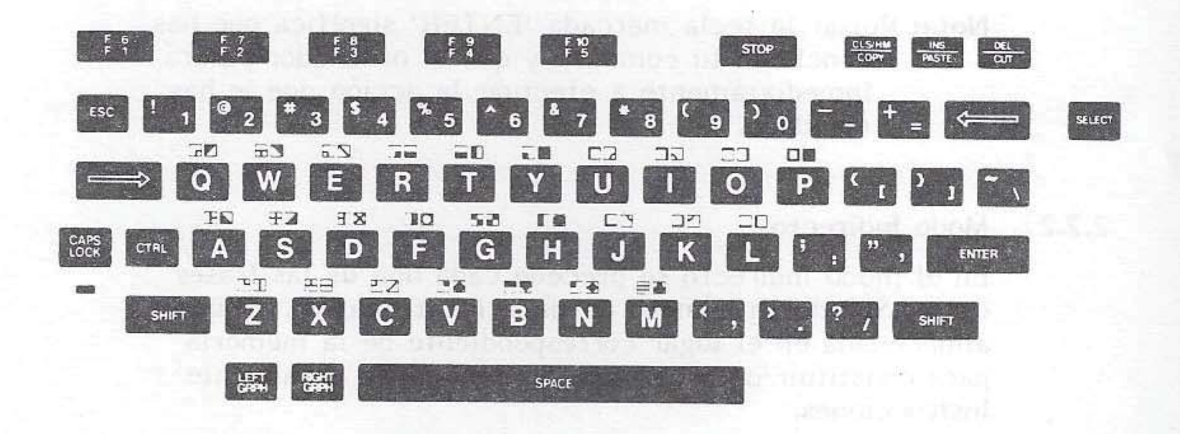
2.3 TECLADO

Generalmente nos comunicamos con el ordenador mediante frases **(comandos e instrucciones)**, enviadas mediante el teclado. Lo tecleado es transferido al ordenador que lo 'repita' en la pantalla para que vayamos comprobando si nos hemos equivocado o no. Y solamente empieza a tener efecto cuando pulsamos la tecla marcada ENTER (ADENTRO).

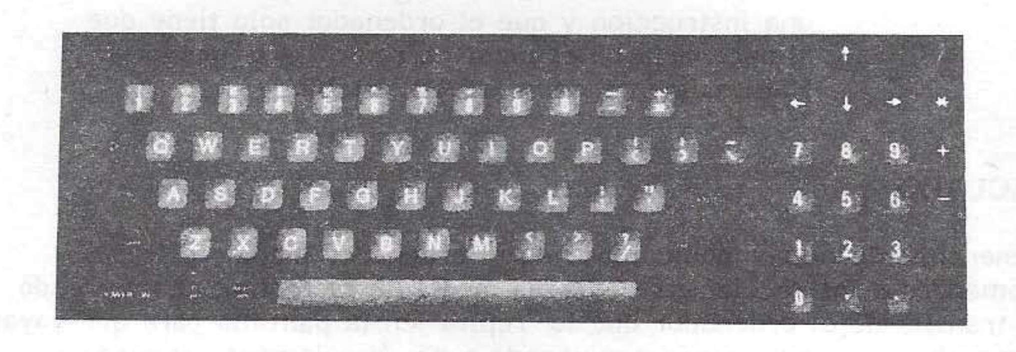


El teclado del ordenador semeja el de una máquina de escribir. Sin embargo, contiene teclas adicionales que son necesarias para comunicarnos más rápida y eficazmente con el ordenador.

El teclado del SV318



El teclado del SV328





Básicamente, podemos dividir el teclado en cinco áreas generales:

* Las teclas **programables como comandos**, rotuladas de F1 a F10, y situadas a lo largo de la fila superior del teclado.

* Las teclas **tipográficas** que ocupan la parte central del teclado. En ellas se encuentran las teclas standard de una máquina de escribir y en su colocación habitual.

* Las teclas de **control de ejecución** del programa.

* Las teclas de **edición de programas** situadas en la periferia del teclado.

* **Teclas especiales.**

* En la parte derecha del teclado aparece un tablero numérico en el SV328, o la palanca de 8 posiciones en el SV318 para desplazamiento del **cursor** o para ser usado como **"joystick de juegos"**.

2.3.1 Teclas programables

Observa la fila superior del teclado.

Las teclas marcadas F1 a F5 actúan simplemente con la pulsación de la tecla. Las marcadas F6 a F10 actúan al pulsar la tecla en cuestión, simultáneamente con la tecla de TURNO, marcada |SHIFT|.

F 6	F 7	F 8	F 9	F 10
F 1	F 2	F 3	F 4	F 5

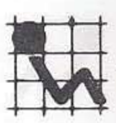


Estas teclas pueden usarse como:

* Comandos de monopulsación: se puede asociar a cada tecla una serie de comandos de manera que con una simple pulsación de la tecla se envíe desde el teclado hacia el ordenador una serie completa de comandos. Para cambiar los valores asociados previamente en fábrica, se debe usar la instrucción KEY que más adelante comentaremos.

* Para suscitar la ejecución de predeterminadas tareas en cualquier momento que se desee interrumpiendo el seguimiento del programa actuante. Para ello, ha de reflejarse en el programa las correspondientes instrucciones condicionales 'perdurativas': ON KEY..., para que CUANDO se pulse una de esas teclas durante la ejecución del programa, se efectúe la tarea mencionada en la instrucción correspondiente.

A la puesta en marcha del ordenador, cada una de estas teclas funcionales tiene prescrito desde fábrica un determinado comando tal y como enumeramos a continuación.



En BASIC para cassette

TECLA	FUNCION PREDEFINIDA	DESCRIPCION
F1	COLOR	Cambia el texto, y los colores del fondo y del marco de la imagen en pantalla.
F2	AUTO |enter|	Para generar automáticamente los números de línea de las instrucciones de un programa.
F3	GOTO	Para comenzar la ejecución del programa residente en memoria a partir de cualquier número de línea.
F4	LIST	Para listar parte o todo el programa residente en memoria.
F5	RUN |enter|	Para 'ejecutar' el programa residente en memoria.
F6	COLOR 15,4,5 |enter|	Expone los caracteres en blanco sobre un fondo azul y con un marco también azul. Son los valores prescritos al poner en marcha el ordenador.
F7	CLOAD"	Para CARGAR desde el cassette un programa. Se puede añadir el nombre del programa que se desea cargar, antes de pulsar |enter|.
F8	CONT |enter|	Para que SIGA la ejecución del programa después de haberle mandado que PARE.
F9	LIST. |enter|	Para que liste la última línea de instrucciones que has metido.
F10	|cls| RUN |enter|	Similar a F5, pero haciendo que previamente a la ejecución del programa se LIMPIE la pantalla.



En BASIC para diskette

TECLA	FUNCION PREDEFINIDA	DESCRIPCION
F1	FILES |enter1|	Para mostrar el 'catálogo' de ficheros grabados en el diskette.
F2	LOAD "1:	Para CARGAR un programa grabado en la unidad de disco número 1. El nombre debe reseñarse inmediatamente después de los dos puntos y antes de pulsar |enter|.
F3	SAVE "1:	Para güardar en el diskette alojado en la unidad de disco 1, el programa residente en memoria. El nombre del programa debe escribirse inmediatamente detrás de los dos puntos, y antes de pulsar |enter|.
F5	RUN |enter|	Para EJECUTAR el programa actualmente residente en memoria.
F6	COLOR 15,4,5 |enter|	Muestra los caracteres en blanco sobre un fondo azul y con un borde de imagen también azul. Estos colores son los prescritos para cuando el ordenador se pone en marcha.
F7	CLOAD"	Para CARGAR un programa desde el cassette. El nombre debe escribirse a continuación.
F8	CONT |enter|	Para hacer que SIGA la ejecución del programa que se obligó a que 'parara'.
F9	LIST. |enter|	Para listar la última línea de instrucciones que has metido.
F10	|cls| RUN |enter|	Similar a F5, excepto que antes de EJECUTAR el programa, hace que se LIMPIE la pantalla.



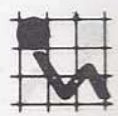
2.3.2 Teclas tipográficas

Esta parte del teclado semeja en todo al teclado standard de las máquinas de escribir. Consta de las siguientes teclas:

* Letras minúsculas y mayúsculas. Lo prescrito como standard es minúscula. Al pulsar la tecla marcada CAPS LOCK se enciende el piloto indicador rojo, y queda el teclado 'enclavado' a mayúsculas. Se suelta este enclavamiento volviendo a pulsar CAPS LOCK una segunda vez. Además puede usarse la tecla de TURNO marcada SHIFT, simultáneamente con cualquier tecla de letra, para generar la mayúscula correspondiente; también se usa la misma tecla SHIFT para elegir el símbolo superior de cualquiera de las teclas.
* Cifras de 0 a 9. Para el ordenador SV328, se dispone adicional y separadamente un tablero sólo numérico para más rapidez cuando se requieran solamente números.
* Símbolos: signos de puntuación, operadores aritméticos y lógicos. El marcado en la parte inferior de la tecla, se obtiene al pulsar simplemente esa tecla. El marcado en la parte superior, se consigue pulsando esa tecla simultáneamente con la tecla de TURNO, marcada SHIFT.



Carácter	Nombre
!	Admiración
@	'Arrobas'
#	Numeral (almohadilla)
$	Dólar
%	Porcentaje
↑	Exponenciación
&	('Ampersand')
*	Asterísco o multiplicación
(	Paréntesis de apertura
)	Paréntesis de cierre
_	Subrayado
-	Guión o signo menos
=	Signo igual o asignación
[	Corchete de apertura
]	Corchete de cierre
{	Llave de apertura
}	Llave de cierre
\	Raya invertida
:	Dos puntos
;	Punto y coma
'	Apóstrofe
,	Coma
.	Punto, punto decimal
<	Menor que
>	Mayor que
/	Raya o división
?	Interrogación



* Barra espaciadora que sirve para dos propósitos:

  (i) Enviar el carácter espacio o blanco.
  (ii) Para **suscitar** la ejecución de una tarea predeterminada en el programa, establecida por una instrucción condicional 'perdurativa': ON STRIG..., en relación con los juegos.

* |shift| Pulsando esta tecla de TURNO, junto con una tecla de letras hace que se envíe esa letra en mayúsculas, y junto con una tecla de símbolos, el marcado en la parte superior.

* |caps lock| Anula la posibilidad de enviar letras en minúscula (sólo letras). Es una tecla 'basculadora' que a cada pulsación cambia de estado permitiendo o evitando las mayúsculas. Se dice que es la tecla de enclavamiento de mayúsculas.

Nota: Esta tecla sirve también para el diagnóstico inicial del sistema; i.e.: cuando se pone en marcha el ordenador, efectúa automáticamente una comprobación funcional del sistema, lo que indica iluminando temporalmente esta tecla. Si el sistema presenta algún fallo, continuará iluminada permanentemente. En esas ocasiones, apaga el ordenador y comprueba todas las conexiones antes de volver a encenderlo otra vez.

2.3.3 Teclas de control de ejecución

Este grupo de teclas se usa para controlar la ejecución de un programa por el ordenador:

|STOP| Pulsando esta tecla, se consigue que el ordenador PARE de ejecutar el programa que estuviera realizando, con lo que puedes dictarle otros comandos o examinar algunos valores. Pulsandola por segunda vez, se le indica que reanude el programa en el punto en que fue 'parado'.

|ENTER| Pulsa esta tecla para indicar que has concluido de teclear un comando o una instrucción. Al pulsarla, el ordenador interpreta que has concluido y que debe **'introducir'** lo que has tecleado y comenzar a examinarlo.

No uses esta tecla cuando llegues al final de una línea de pantalla, porque automáticamente el ordenador continuará en la línea siguiente. Es imprescindible que solamente la pulses cuando realmente has concluido de dictarle un comando o una instrucción. Como veremos, es totalmente equivalente a la marca de 'retorno de carro' preceptuada en el código ASCII.



|CTRL| y |STOP| — La pulsación simultánea de estas dos teclas le indica al ordenador que PARE definitivamente el programa que estaba ejecutando y se coloque en modo de comando.

2.3.4 Teclas de edición

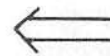
Este grupo de teclas, conjuntamente con las cuatro teclas de desplazamiento del cursor se usan para 'editar' programas (y con editar, queremos decir, revisar, corregir y cambiar las instrucciones).

|CLS/HM / COPY| — Con la función marcada en la parte superior se LIMPIA la pantalla y se mueve el cursor a la posición de BASE (HOME) que es la esquina superior izquierda de la pantalla. La función marcada en la parte inferior, hace que se COPIE en memoria el texto que aparece en la pantalla.

|INS / PASTE| — La función superior sirve para INSERTAR caracteres dentro de una línea. Para ello desplaza el cursor hasta el sitio donde deseas insertar un carácter, pulsa esta tecla y escribe el texto que deseas insertar. La función inferior hace que en lugar de insertar texto, se SUSTITUYA el que aparece en imagen por el que tú teclees.

|DEL / CUT| — La función superior se usa para SUPRIMIR el carácter situado encima del cursor. La función inferior sirve para cortar una línea de instrucciones en dos.

|ESC| — Esta tecla se usa para ESCAPAR del control de un determinado programa. Su función habitual es interrumpir la ejecución de un programa o continuar la operación después de una interrupción. Dentro de los programas, se usa para que los caracteres enviados a continuación de ESCape, no correspondan a su significado habitual (primordial al trabajar con impresoras).


Esta tecla no se usa mucho en BASIC. Se emplea en los procesadores de textos o programas de aplicación similares para hacer avanzar de golpe cinco espacio y comenzar un párrafo.

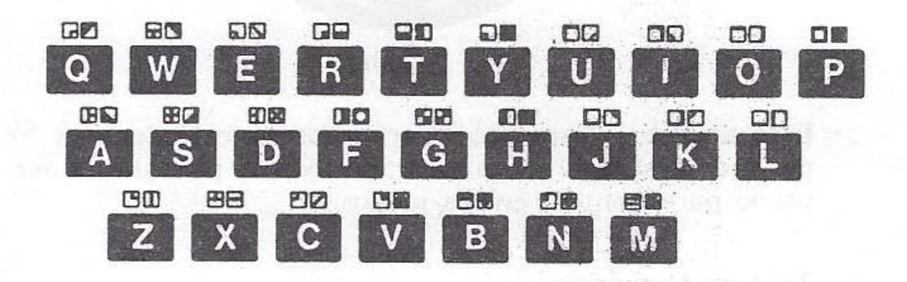
⟸ Esta tecla tampoco se usa mucho en BASIC. Retrocede el cursor una posición y suprime el carácter situado inmediatamente a la izquierda del cursor (antes de pulsar la tecla).



2.3.5 Teclas especiales

|LEFT GRPH| |RIGHT GRPH|

Estas teclas se utilizan alternativamente para elegir los símbolos GRAFICOS situados a IZQUIERDA y DERECHA respectivamente de cada una de las teclas alfabéticas. Si pulsas la tecla |LEFT GRPH| y la mantienes pulsada mientras pulsas una de las teclas simbólicas, enviarás hacia el ordenador y será 'repicado' en pantalla el símbolo situado por encima y a la izquierda de la correspondiente tecla pulsada. Pulsando alternativamente la tecla |RIGHT GRPH| se elige símbolo situado a la derecha.



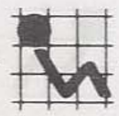
|SELECT| La tecla SELECT está presente en el ordenador SV318 y en el SV328 y permiten ser aprovechadas primordialmente en programas de aplicación procesadores de texto y adquisición de datos; pero no tiene mucha aplicación en los programas en BASIC.

|PRINT| Sólo presente en el ordenador SV328, se le aplica lo anteriormente dicho para la tecla SELECT.

2.3.6 Teclas de desplazamiento del cursor

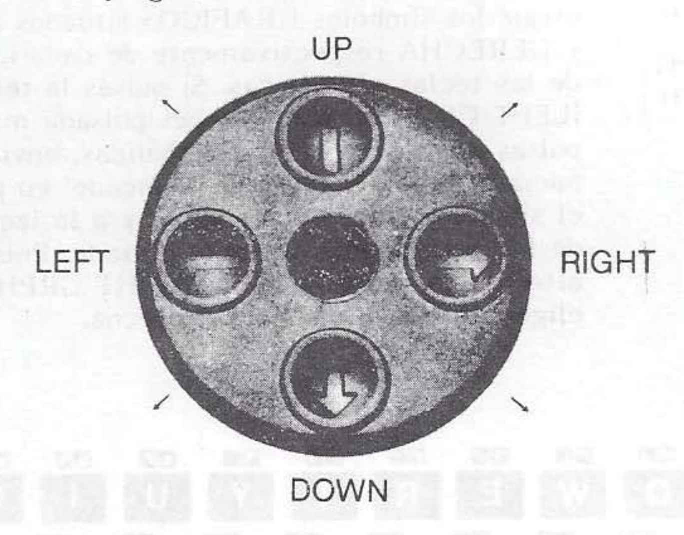
Las teclas marcadas con flechas (arriba, abajo, izquierda, derecha) controlan el movimiento del cursor por toda la pantalla. Pulsando simultáneamente la tecla con flecha ascendente y a izquierda, harás que el cursor se desplace hacia la esquina superior izquierda de la pantalla. Las otras combinaciones trabajan de la misma manera, consiguiendo en total 8 direcciones para el desplazamiento del cursor.

Flecha a izquierda (←). Si el cursor llega al borde izquierdo de la imagen, continuará en la parte derecha de la línea precedente.



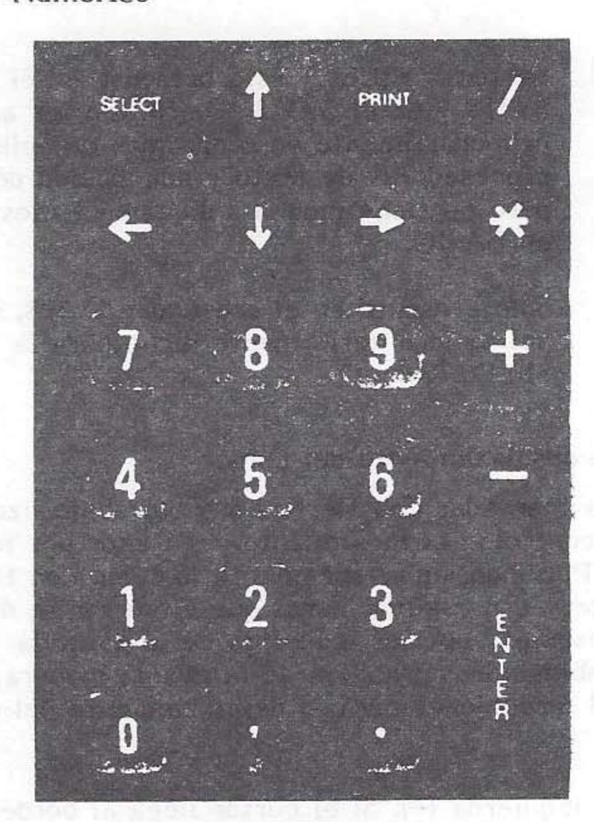
Flecha a derecha, (→). Si el cursor llega al margen derecho de la imagen, continuará por el margen izquierdo de la línea siguiente.

2.3.7 Joystick de juegos/control del cursor



Este accesorio especial incorporado en el ordenador SV318 permite desplazar el cursor por toda la pantalla, y ser usado para control en los juegos.

Tablero Numérico



Este accesorio incorporado en el ordenador SV328, agrupa las cifras y las operaciones matemáticas más usuales (+ – * /) para mayor facilidad al tratar con fórmulas numéricas o para efectuar cálculos rápidos como si se tratara de una calculadora. También dispone de las flechas indicativas del desplazamiento del cursor.



INFORMACION GENERAL SOBRE LA PROGRAMACION EN BASIC

3.1 INTRODUCCION

Para controlar un ordenador, deben dársele los mandatos (comandos e instrucciones) en el lenguaje que la máquina comprende. Los ordenadores SVI comprenden un lenguaje llamado Microsoft BASIC. La palabra BASIC proviene de "Beginner's All Purpose Symbolic Instruction Code", y realmente es un subconjunto de frases encabezadas por **palabras clave**, en ortografía inglesa, que indican al ordenador el tipo de acción que queremos realice.

El BASIC de Microsoft es una versión ampliada del BASIC standard Microsoft 5.3, que incluye facilidades para gráficos, música, y diversos equipos periféricos conectados al ordenador personal. Hablando en general, BASIC está diseñado para seguir el GW BASIC que es el standard en las máquinas de **16 bits**. Sin embargo, se ha empleado gran esfuerzo en hacer que el sistema sea tan flexible y ampliable como sea posible.

Además el Microsoft BASIC está caracterizado por una precisión de 14 dígitos en las funciones aritméticas. Lo que significa que ya no se generarán los extraños errores de redondeo que confunden a los usuarios principiantes. Cada una de las funciones inherentes del BASIC Microsoft se calcula con esta doble precisión.

3.2 ¿QUE ES LA PROGRAMACION?

Programar es el arte de escribir las instrucciones y la información que queremos que el ordenador examine, interprete, y procese. Los programas difieren unos de otros en las instrucciones que los componen y en el orden en que se colocan. Además, los programas escritos de acuerdo con las reglas sintácticas de un lenguaje no corresponden con las instrucciones y estructuras que los programas han de tener en otro lenguaje informático.

Hay dos formas diferentes de mandarle ejecutar acciones al ordenador: en el modo directo, que usualmente llamamos **comandos.** En el modo indirecto, que usualmente llamamos instrucciones.

En ambos modos se pueden dar los mandatos uno a uno, o varios sucesivos, formando una **línea de comandos** o una línea de instrucciones. La única diferencia estriba en que las líneas de instrucciones deben estar precedidas por un número identificativo de la línea que además sirve para marcar el orden en que tendrán que ejecutarse.



3.3 FORMATO DE LAS LINEAS DE INSTRUCCIONES

Un programa en BASIC consta de una serie de **líneas** con el siguiente formato:

nnnnn instrucción BASIC [:instrucción BASIC] .... ['comentario]

|enter|

Una línea de instrucciones siempre comienza con un número de línea nnnnn, que ha de ser entero y estar en la gama 0 a 65529. Los números de línea además de identificar la línea en cuestión sirven para determinar el orden en que se guardan las instrucciones en la memoria del ordenador, y por tanto el orden en que será ejecutado el programa.

La línea puede contener hasta un máximo de 255 caracteres y constar de más de una instrucción (líneas multi-instrucción) separando cada una de la anterior por medio del signo dos puntos (:).

Se pueden añadir en el extremo de la línea los comentarios que se deseen separándolos simplemente por el signo apóstrofe (').

Toda línea de programa debe concluirse pulsando la tecla |ENTER|. Eso es lo que determina el final de una línea del programa desde el punto de vista **logical** (aunque desde el punto de vista **fisical** puede ocupar varias líneas o renglones en pantalla).

3.4 REPERTORIO DE CARACTERES

Todas las frases empleadas en un programa han de estar formadas por los símbolos habituales: letras, cifras, signos de puntuación u operación, y de los caracteres gráficos definidos en el sistema.

En las letras pueden usarse tanto mayúsculas como minúsculas.

Las cifras permitidas son los 10 **dígitos**, del 0 al 9.

Los signos pertenecientes al repertorio son:

Signo	Significado
=	Igual o símbolo de asignación
+	Signo más
-	Signo menos
*	Asterísco o multiplicación
/	Barra inclinada o división
^	Exponenciación.



(	Paréntesis de apertura
)	Paréntesis de cierre
%	Porcentaje
#	Diésis numérico (almohadilla)
$	Dólar
!	Admiración
[	Corchete de apertura
]	Corchete de cierre
{	Llave de apertura
}	Llave de cierre
,	Coma
.	Punto o punto decimal
'	Apóstrofe
;	Punto y coma
:	Dos puntos
&	Concatenación ('ampersand')
?	Interrogación
<	Menor que
>	Mayor que
\	Barra o símbolo de división entera
@	'Arrobas'
_	Subrayado
-	Guión (suprimir el último carácter tecleado)
ESC	Escape
⟹	Colocar el cursor en el siguiente tope de tabulación (están fijados cada 8 columnas).
|ENTER|	Concluye un comando o instrucción.



3.5 PALABRAS RESERVADAS

En el lenguaje BASIC hay un conjunto prefijado de palabras que son **reservadas** en el lenguaje, y que se denominan habitualmente **clave.** No pueden usarse con ningún otro significado distinto al preceptuado en las reglas sintácticas del lenguaje. Por lo tanto, si intentas usar cualquiera de las palabras enunciadas a continuación como nombre de una variable, el ordenador señalará el error correspondiente.

ABS	ERL	LOG	RENUM
AND	ERR	LPOS	RESTORE
APPEND	ERROR	LPRINT	RESUME
ASC	EXP	LPRINT USING	RETURN
ATN	FIELD	LSET	RIGHT$
ATTR$	FILES	MAXFILES	RND
AUTO	FIX	MERGE	RSET
BEEP	FOR	MID$	RUN
BIN$	FPOS	MKI$	SAVE
BLOAD	FRE	MKD$	SCREEN
BSAVE	GET	MKS$	SET
CALL	GOSUB	MOD	SGN
CDBL	GOTO	MOTOR ON	SIN
CHR$	HEX$	MOTOR OFF	SOUND
CINT	IF	NAME	SPACE$
CIRCLE	INKEY$	NEW	SPC
CLEAR	INP	NEXT	SPRITE OFF
CLICK	INPUT	NOT	SPRITE ON
CLOAD	INPUT#	OCT$	SPRITE STOP
CLOAD?	INPUT$	ON ERROR GOTO	SPRITE$
CLOSE	INSTR	ON GOSUB	SQR
CLS	INT	ON GOTO	STEP
COLOR	INTERVAL	ON INTERVAL GOSUB	STICK
CONT	INTERVAL OFF	ON KEY GOSUB	STOP
COS	INTERVAL ON	ON SPRITE GOSUB	STOP ON
CSAVE	INTERVAL STOP	ON STOP GOSUB	STOP OFF
CSNG	IPL	ON STRIG GOSUB	STOP STOP
CSRLIN	KEY	OPEN	STRIG
CVI	KEY LIST	OR	STRIG OFF
CVS	KEY ON	OUT	STRIG ON
DATA	KEY STOP	OUTPUT	STRIG STOP
DEFDBL	KEY (n) OFF	PAD	STR$
DEFFN	KEY (n) ON	PAINT	STRING$
DEFINT	KEY (n) STOP	PEEK	SWAP
DEFSNG	KILL	PLAY	TAB
DEFSTR	LEFT$	POINT	TAN
DEFUSR	LEN	POKE	THEN
DELETE	LET	POS	TIME
DIM	LFILES	PRESET	TROFF
DRAW	LINE	PRINT	TRON
DSKI$	LINE INPUT#	PRINT USING	USR
DSKO$	LIST	PRINT#	VAL
ELSE	LLIST	PRINT# USING	VARPTR
END	LOAD	PSET	VPEEK
EOF	LOC	PUT SPRITE	VPOKE
EQV	LOCATE	READ	WAIT
ERASE	LOF	REM	WIDTH
			XOR



3.6 CONSTANTES

Denominamos constantes a los valores de los datos usados durante la ejecución. En BASIC se consideran dos clases de constantes: constantes de índole **literal,** y constantes de índole **numeral.**

Se dice que una constante es de índole literal cuando debe ser considerada por el ordenador como una secuencia de caracteres del repertorio admitido. Se encierran entre comillas y puede haber hasta 255 caracteres cualesquiera del repertorio. Por ejemplo:

"$25.000,00"
"HOLA"
"Número de Empleados"

Las constantes de índole numérica son las que representan cantidades positivas o negativas. Se distinguen varias subclases:

1. Enteras — Números enteros en la gama -32768 a +32767.

2. Coma constante — Números reales positivos o negativos (i.e. son los números que contienen una parte fraccionaria, señalada por un punto -no por la coma-).

3. Coma flotante — Números positivos o negativos representados en la llamada notación científica. Una constante en coma flotante consta de una parte entera o de coma constante (la mantisa) seguida de la letra E y de un número entero (el exponente en base 10). La gama permitida para las constantes en coma flotante es en valor absoluto de 10E-38 a 10E+38.

Por ejemplo:
235.988E-7 = .0000235988
2359E6 = 2359000000

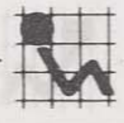
(Las constantes en coma flotante de doble precisión usan la letra D en lugar de la letra E).

4. Hex — Números hexadecimales (en base 16) con el prefijo &H.

Por ejemplo:
&H76
&H32F

5. Octal — Números octales (en base 8) mediante el prefijo &O.

Por ejemplo:
&O347



6. Binarios — Números binarios (en base 2) designados por el prefijo &B.

Por ejemplo:
&B11100111

3.6.1 Simple y doble precisión en las constantes numéricas

Las constantes numéricas pueden ser bien de simple precisión o de doble precisión. Cuando la constante numérica es de simple precisión, se maneja internamente con 7 dígitos, aunque al exponerla sólo se muestran 6. Las constantes de doble precisión se manipulan internamente con 14 dígitos y se muestran externamente con 14 dígitos. La doble precisión es el modo prescrito para omisiones.

Se considera una constante de simple precisión a cualquier constante numérica que tenga una de las siguientes características:

1. Notación científica usando la letra E.
2. Un signo de admiración trasero (!).

Por ejemplo: -1.09E-06
22.5!

Una constante de doble precisión es cualquier constante numérica que posea una de las siguientes características:

1. Cualquier número de dígitos sin parte exponencial ni designador de tipo.
2. Notación científica usando la letra D.
3. Un signo diésis trasero (#).

Por ejemplo: 3489
345692811
7654321.1234
-1.09432D-06
3489.0#

3.7 VARIABLES

Variables son los nombres usados para identificar los datos en un programa BASIC; y los valores de las variables son con los que se opera en el programa. El valor de una variable puede ser asignado explícitamente por el programador, directamente o como resultado de un cálculo. Antes de asignarle un valor a una variable, se presupone que es 0.



3.7.1 Nombres de variables y designador de la índole de sus valores

Los nombres de variables en BASIC pueden tener cualquier longitud aunque sólo los 2 caracteres anteriores son significativos. El nombre puede contener letras y números, aunque el primer carácter debe ser siempre una letra. También está permitido usar signos especiales para designar la índole del dato.

Un nombre de variable no debe tener ni constar de palabras reservadas. Las palabras reservadas incluyen todos los verbos BASIC, nombres de funciones y nombres de operadores. Si la variable comienza con las letras FN, se supone que corresponde a una función definida por el usuario.

Las variables pueden tener valores de índole numérica o valores de índole litérica, denominándose correspondientemente variables numéricas o numerales y variables litéricas o literales.

Las variables literales son aquéllas cuyos nombres terminan con el signo dólar. Por ejemplo:

A$ = "INFORME DE VENTAS"

El signo dólar es el que designa la clase o índole de la variable. En este caso declara que los valores son considerados como 'cadenas' de caracteres, que deben ser tomados "al pie de la letra".

Los nombres de las variables numéricas, pueden también incluir un designador de la clase a que pertenecen, tal y como sigue:

%	Variable entera
!	Variable de simple precisión
#	Variable de doble precisión

Si se omite el símbolo designador, está prescrito que se considere como una variable numérica de doble precisión.

Ejemplos de nombres de variables en BASIC:

PI#	Declara una variable con valores en doble precisión.
MINIMO!	Declara una variable con valores de simple precisión.
LIMIT%	Declara una variable con valores enteros
N$	Declara una variable con valores literales
ABC	Representa una variable con valores de doble precisión.

Además, se pueden incluir en un programa las instrucciones DEFINT, DEFSTR, DEFSNG y DEFDBL para designar la clase correspondiente a determinados nombres de variables.



3.7.2 Multivariables

Una **matriz** es un grupo de datos homogéneos dispuestos en forma de **tabla** a los que se hace referencia usando un solo nombre de variable. Cada valor individual de la **tabla** se denomina un elemento, y se selecciona añadiendo un **sufijo** numérico al nombre de la multivariable.

Una tabla necesita ser definida y dimensionada. Definirla significa designar el nombre de la multivariable y la clase de variables que la compone. Dimensionarla significa fijar el número de elementos que contiene y la colocación adoptada para la tabla. El número máximo de dimensiones para una tabla es 255. Por ejemplo, con la instrucción:

DIM A$(3)

comunicamos al ordenador la existencia de una ringla **monodimensional** designada A$. Todos sus elementos son pues variables literales y en el momento de ejecutar esta instrucción, el ordenador ocupa 'espacio' para registrar cuatro valores literales que inicialmente son espacios en blanco. Podemos imaginarnos con bastante exactitud que en la memoria habrá cuatro 'cajetines' denominados:

A$ (0)
A$ (1)
A$ (2)
A$ (3)

Para poder identificar cada uno de los cuatro elementos de esta tabla.

Podemos crear ahora una tabla bi-dimensional, designada con la letra B, que esté formada por variables numéricas de simple precisión. Todos los elementos inicialmente estarán puestos a cero. Usaremos la instrucción:

DIM B (1,2)

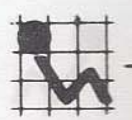
y el ordenador ocupará espacio en memoria para seis elementos dispuestos regularmente como

B (0,0)	B (0,1)	B (0,2)
B (1,0)	B (1,1)	B (1,2)

El elemento colocado en la segunda fila, primera columna se designará B (1,0).

Si se menciona en un programa un elemento de una tabla, sin haber designado previamente dicha tabla, el sistema presupondrá que se trata de una tabla monodimensional con 11 elementos (sufijos 0 a 10).

* Array, en inglés, es un conjunto ordenado de forma **regular** (aquí matriz no tiene nada en común con las auténticas "matrices" matemáticas). 'Arreglos' en Sudamérica, y para nosotros 'ringla'.



3.7.3 Requisitos de espacio

La siguiente tabla presenta el número de **octetos** ocupados por los valores correspondientes a las clases de variables.

Variables	Clase	Octetos
	Entera	2
	Simple precisión	4
	Doble precisión	8

Tablas	Clase	Octetos
	Entera	2
	Simpre precisión	4
	Doble precisión	8

Cadenas	3 octetos suplementarios más el número de caracteres que componen la cadena.

3.8 CONVERSION DE CLASES

Una constante numérica puede convertirse de una clase a otra teniendo en cuenta las siguientes reglas.

1. Si una variable numérica de una clase se le asigna como valor una constante de otra clase, el número será registrado como corresponde a la clase designada por el nombre de la variable. Por ejemplo:

```
10 A%=23.42
20 PRINT A%
RUN
23
Ok
```

La línea 10 especifica que la variable A es entera. Luego le asigna la constante 23.42 y por tanto, tiene prioridad la clase designada por el nombre de la variable.

2. Cuando se evalúa una expresión, sus operandos se convierten todos a una clase homogenea, y se elige siempre la clase con mayor precisión. Esto sucede tanto para operaciones aritméticas como relacionales. Además, el resultado de las expresiones aritméticas se entrega con esa misma precisión. Por ejemplo:

```
10 D=6/7
20 PRINT D
RUN
.857142857142 86
Ok
```

El cálculo anterior se realizó en doble precisión y el resultado también se presentó como un valor en doble precisión.



```
10 D!=6/7
20 PRINT D!
RUN
.857143
Ok
```

El cálculo se realizó en doble precisión y el resultado se redondeó a un valor de simple precisión.

3. Los operadores lógicos convierten sus operandos a valores enteros y entregan un resultado también entero. Los operandos deben estar en la gama -32768 a +32767, para que no se produzca un error de **sobrepasamiento**.

4. Cuando un valor en coma flotante se convierte a entero, simplemente se prescinde de la parte fraccionaria ('se trunca'). Por ejemplo:

```
10 C%=55.88
20 PRINT C%
RUN
55
Ok
```

5. Si un valor en doble precisión es asignado a una variable de simple precisión, sólo serán válidos los primeros 6 dígitos del número convertido; dado que únicamente se emplean 6 dígitos en los valores de simple precisión. Por ejemplo:

```
10 A!=SQR(2)
20 B=A!
30 PRINT A!,B
RUN
 1.41421            1.41421
Ok
```

3.9 EXPRESIONES Y OPERADORES

Las expresiones pueden ser también literales o numerales. Una expresión numeral puede estar formada por una constante numérica, por una variable numérica o por una combinación de constantes y variablesscon operadores que al ser evaluada produce un resultado numérico.

Los operadores son signos que señalan las operaciones matemáticas o lógicas a realizar sobre los datos, que se denominan **operandos.** Los operadores admitidos en BASIC se dividen en cuatro categorías:

1. Aritméticos
2. Relacionales
3. Lógicos
4. Funcionales



3.9.1 Operadores Aritméticos

Los operadores aritméticos, en orden de **prioridad de la operación**, son:

Operador	Operación	Ejemplo
^	Potenciación	X^Y
-	Negación	-X
*,/	Multiplicación, División en coma flotante	X*Y X/Y
+,-	Suma, resta	X+Y X-Y

Se usan paréntesis en la forma habitual para cambiar este orden de prioridad. Las operaciones reflejadas entre paréntesis tienen la máxima prioridad, y se realizan en primer lugar. Dentro de los paréntesis se mantienen las prioridades establecidas anteriormente.

3.9.1.1 Divisón entera y aritmética de congruencias

Se dispone de dos operadores adicionales en BASIC: La división entera se designa mediante el símbolo \. En una división entera, los operandos se 'truncan' a enteros en la gama -32768 a +32767 antes de efectuar la operación, y asimismo, el cociente se trunca para conseguir el resultado. Por ejemplo:

```
PRINT 10\4
2
Ok
PRINT 25.68\6.99
4
Ok
```

La división entera (también llamada división restos módulo) el divisor va detrás de la multiplicación y la división en coma flotante en cuanto a prioridad.

Las **congruencias** (o aritmética modular) usan el operador **MOD** para obtener el resto de una división entera como un valor también entero. Por ejemplo:

```
PRINT 10.4 MOD 4         10\4=2 tiene un resto de 2
2
Ok
PRINT 25.68 MOD 6.991    25\6=4 tiene un resto de 1
1
Ok
```

En el primer ejemplo, diriamos que 10 es congruente con 2 módulo 4, porque ambos tienen el mismo resto.



Esta operación en cuanto a orden de preferencia o prioridad, viene detrás de la división entera.

3.9.1.2 Sobrepasamiento (desborde) y división por cero

Si al evaluar una expresión, se encuentra un valor que sobrepasa la gama permitida (si es enteros la gama -32768 a +32767) se produce un error llamado de desbordamiento, y se presenta en pantalla habitualmente, el mensaje **OVERFLOW.** Se interrumpe la ejecución del programa.

Asimismo, si en una evaluación se encuentra la operación de dividir por cero, se mostrará el mensaje de error correspondiente y se detendrá la ejecución del programa.

3.9.2 Operadores Relacionales

Los operadores **de relación** se usan para comparar dos valores. El resultado de la comparación es un número entero, de valor -1 cuando la comparación es **cierta**, y de valor 0 cuando es **falsa.** Este resultado puede examinarse para posteriormente elegir una alternativa de acción en el programa (véase descripción de las sentencias condicionales IF).

Los operadores de relación son:

Operador	Relación examinada	Ejemplo
=	Igualdad	X=Y
<> o ><	Desigualdad	X<> Y
<	Menor que	X < Y
>	Mayor que	X > Y
< = o =>	Menor o igual que	X< =Y
>= o =<	Mayor o igual que	X> =Y

El signo igual, empleado como operador de relación también se usa como **operador de asignación** para asociar un valor a una variable.

Cuando en una expresión se combinan operadores aritméticos y relacionales, siempre se efectúan primero las operaciones aritméticas, por ejemplo, la expresión:

X+Y< (T-1)/Z

dará como resultado -1 (cierto) si el valor de X+Y, es menor que el valor de T-1 dividido por Z.



Más ejemplos:

```
IF SIN(X)< 0 GOTO 1000
IF I MOD J<> 0 THEN K=K+1
```

3.9.3 Operadores Lógicos

Los operadores lógicos efectúan comprobaciones en relaciones múltiples, manipulación bit a bit u operaciones **booleanas.** El operador lógico entrega un resultado **calibrado en binario,** que sólo puede ser cierto (distinto de 0) o falso (cero). En una expresión, las operaciones lógicas se efectúan después de las operaciones aritméticas y relacionales. El valor resultante de una operación lógica está determinado por los operandos, tal y como se muestra en la tabla 1, en la que se presentan las operaciones lógicas según su orden de prioridad.

Tabla 3.1 Tablas de verdad para operadores lógicos BASIC

NOT	X	NOT X
	1	0
	0	1

AND	X	Y	X AND Y
	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	X	Y	X OR Y
	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR	X	Y	X XOR Y
	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV	X	Y	X EQV Y
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP	X	Y	X IMP Y
	1	1	1
	1	0	0
	0	1	1
	0	0	1



Los operadores lógicos pueden conectar dos o más relaciones y entregar un valor cierto o falso que será usado en una decisión.

Ejemplo:
```
IF D < 200 AND F < 4 THEN 80
IF I > 10 OR K < 0 THEN 50
IF NOT (P = -1) THEN 100
```

Los operadores lógicos convierten sus operandos que han de ser números enteros en la gama -32768 a +32767, a su equivalente binario con 16 bits, con signo y en complemento a doses. Luego efectúan la operación designada aplicando los operadores a los bits **homólogos** de los operandos, determinando así el bit correspondiente del resultado. Antes de presentar el resultado, vuelve a calcularse el equivalente en base 10.

De esta manera, es posible usar los operadores lógicos para comprobar si ciertos octetos responden a un determinado **perfil binario.** Por ejemplo, el operador AND puede usarse para **anular** todos menos 1 de los bits de un octeto que indique el estado de un portal de entrada/salida de la máquina. El operador OR puede usarse para **aunar** dos octetos y crear un valor binario particular. El siguiente ejemplo te ayudará a comprobar cómo funcionan los operadores lógicos.

63 AND 16 = 16	63	0011	1111
	16	0001	0000
	63 AND 16	0001	0000
15 AND 14 = 14	15	0000	1111
	14	0000	1110
	15 AND 14	0000	1110
-1 AND 8 = 8	-1	1111	1111
	8	0000	1000
	-1 AND 8	0000	1000
4 OR 2 = 6	4	0000	0100
	2	0000	0010
	4 OR 2	0000	0110
10 OR 10 = 10	10	0000	1010
	10	0000	1010
	10 OR 10	0000	1010
-1 OR -2 = -1	-1	1111	1111
	-2	1111	1110
	-1 OR -2	1111	1111

TWOSCOMP = (NOTX) +1 — El **complemento a doses** de cualquier entero se obtiene negando cada uno de los bits y sumándole 1.



3.9.4 Operadores Funcionales

Una función se usa en una expresión para designar una operación prefijada que ha de ser efectuada sobre el operando de la función, que se denomina **argumento.** BASIC tiene funciones intrínsecas que residen en el propio sistema, tales como la raíz cuadrada **SQR**, o el seno de un ángulo **SIN.**

El BASIC también permite que el usuario defina sus propias funciones, mediante la instrucción DEF FN.

3.9.5 Operaciones con literales

Se pueden **concatenar** cadenas de caracteres usando el operador (+).

Ejemplo:

```
10 A$="FILE" : B$="NAME"
20 PRINT A$+B$
30 PRINT "NEW "+A$+B$
RUN
FILENAME
NEW FILENAME
Ok
```

También se pueden comparar variables y constantes literales usando los mismos operadores de relación que se usan con los números.

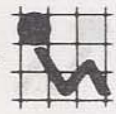
=   <>   <   >   <=   >=

Las comparaciones entre literales se hacen tomando un carácter cada vez a partir de la izquierda, y comparando el **código ASCII** correspondiente. Si todos los códigos ASCII son los mismos, las cadenas de caracteres son iguales. En caso contrario, se determina la precedencia de uno u otro carácter según el código ASCII. Si durante la comparación entre literales se alcanza el final de uno de ellos, la cadena más corta a igualdad de tipo de caracteres, se dice que es la más pequeña o que va antes. Los blancos tanto delanteros como traseros, son tenidos siempre en cuenta.

Ejemplos:

```
"AA"<"AB""
"FILENAME"="FILENAME"
"X&">"X#"
"CL ">"CL"
"Kg">"KG"
"SMYTH"<"SMYTHE"
B$<"9/12/83"        siendo B$="8/12/83"
```

Las comparaciones entre cadenas de caracteres pueden usarse para comprobar valores de variables literales o para la ordenación alfabética. Todas las constantes literales usadas en las expresiones comparativas, deben estar encerradas entre comillas.



Recuerda que el número máximo de caracteres permitidos en una línea de programa, incluyendo el número de línea, es de 255.

3.10.2 Editando un programa

Usa el comando LIST para mostrar todo el programa, o parte de él, en la pantalla de modo que puedas **editarlo.** El texto puede entonces modificarse desplazando el cursor hasta el sitio donde necesitas hacer el cambio y efectuando una de las siguientes acciones:

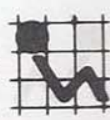
1. Tecleando encima de los caracteres existentes.
2. Suprimiendo los caracteres a la derecha del cursor.
3. Suprimiendo los caracteres a la izquierda del cursor.
4. Insertando caracteres
5. Empalmando caracteres al final de la línea del programa.

Estas acciones se efectúan mediante teclas especiales asociadas a las diversas funciones del editor de pantalla completa (véase la siguiente sección).

Los cambios en una línea quedan registrados cuando se pulsa |ENTER|, pudiendo estar situado el cursor en cualquier parte de la línea. Todos los cambios para esa línea de programa son registrados, y no importa cuántos renglones se vean afectados.

3.10.3 Editor de pantalla completa

La tabla siguiente presenta los códigos hexadecimales para los caracteres de control BASIC y resume sus funciones. La secuencia Control-tecla, normalmente asignada a cada función también se presenta en dicha tabla. Están conformes en todo lo posible con los convenios standard ASCII.



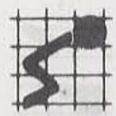


Tabla 3.2 Funciones de control en el BASIC SV
El carácter de control ASCII se consigue pulsando simultáneamente la tecla Control y la tecla designada.

Código HEX	Control y tecla	Tecla especial	Función	Observaciones
01	A		Ignorado	
02	B		Mueve el cursor al principio de la palabra anterior.	La palabra anterior está definida como el siguiente carácter distinto de blanco que aparezca a la izquierda del cursor.
03	C		Interrupción cuando BASIC está esperando respuesta por teclado.	Hace regresar al BASIC al modo directo, sin registrar los cambios que se estuvieran haciendo en la línea editada en ese momento.
04	D		Ignorado	
05	E		Cercena el resto de la línea (limpia el texto hasta el final de la línea del programa).	Mueve el cursor hasta el final de la línea logical, suprimiendo los caracteres sobre los que pasa. Los caracteres tecleados a partir de la nueva posición del cursor se empalman a la línea.
06	F		Mueve el cursor al principio de la siguiente palabra.	La siguiente palabra está definida por el primer carácter distinto de blanco que haya a la derecha del cursor.
07	G		Pitido	Genera un breve pitido.
08	H		Retrocede el cursor, suprimiendo los caracteres sobre los que pasa.	Suprime el carácter a la izquierda del cursor. Todos los situados a la derecha del cursor, se desplazan una posición a la izquierda. Los caracteres subsiguientes y los renglones incluidos en la línea del programa corriente también se desplazan.

09	I		Mueve el cursor al sigueinte tope de TABulación.	Mueve el cursor hasta el siguiente tope de tabulación, colocados cada 8 caracteres, y rellena de espacios en blanco las posiciones pertinentes.
0A	J		Avance de línea.	
0B	K	CLS/HM	Mueve el cursor a la posición base (HOME).	Mueve el cursor a la esquina superior izquierda de la pantalla, pero sin limpiarla.
0C	L	CLS	Limpia pantalla.	Mueve el cursor a la posición base y limpia toda la pantalla, sin tener en cuenta dónde está situado el cursor cuando se pulsa la tecla.
0D	M	ENTER	'Adentro' (retorno de carro).	Marca el final de una línea lógica y el BASIC regresa al modo directo.
0E	N		Empalma al final de la línea.	Mueve el cursor hasta el final de la línea, sin suprimir los caracteres sobre los que pasa. Todos los caracteres tecleados a partir de esta nueva posición y hasta que se pulse |ENTER| son añadidos a la línea del programa.
0F	O		Ignorado	
10	P		Ignorado	
11	Q		Ignorado	
12	R	INS	Basculador de inserción sobre escritura.	Es un conmutador por basculación del modo de inserción. Cuando está activo el modo de inserción, el tamaño del cursor se reduce y los caracteres tecleados se insertan a partir de la posición corriente del cursor. Los caracteres situados a la derecha del cursor se desplazan a medida que se van insertando los nuevos. Automáticamente desplazará a las líneas siguientes cuando sea preciso.
13	S		Ignorado	



14	T		Ignorado	
15	U		Limpia línea logical.	Cuando se pulsan estas teclas, estando el cursor en cualquier posición dentro de la línea NO se registra nada de la línea del programa.
16	V		Ignorado	
17	W	SELECT	Ignorado	
18	X		Ignorado	
19	Y		Ignorado	
1A	Z		Ignorado	
1B	[	ESC	Ignorado	
1C	\		Cursor a derecha.	Mueve el cursor una posición a la derecha.
1D	]		Cursor a izquierda.	Mueve el cursor una posición a la izquierda.
1E	^		Cursor arriba.	Mueve el cursor un renglón hacia arriba, manteniendo la columna.
1F	_		Cursor abajo.	Mueve el cursor un renglón hacia abajo, manteniendo la columna.
7F	DEL	DEL	Suprime	Suprime el carácter situado encima del cursor.

Normalmente, una línea de programa consta de todos los caracteres incluidos en los renglones pertinentes. Durante la ejecución de una instrucción INPUT o de una LINE INPUT, se altera sin embargo, ligeramente esta modificación para aceptar datos tomados de impresos. La línea de programa está restringida a los caracteres realmente tecleados, o aquéllos sobre los que pasa por encima el cursor. En el modo de inserción y con la función de supresión, sólo se mueven los caracteres dentro de una línea de programa.

En el modo de inserción se incrementa la línea de programa excepto cuando los caracteres desplazados escribirán sobre caracteres no blancos que están en la misma línea de programa pero que no forman parte de la línea de programa. En este caso, los caracteres no blancos, al no ser parte de la línea de programa se conservan, y los caracteres al final de la línea de programa se desechan. Así se preservan las etiquetas que existieran antes de la instrucción INPUT. Si se teclea un carácter incorrecto como parte de la línea vigente, puede ser suprimido mediante la tecla de retroceso de la línea vigente, puede ser suprimido mediante la tecla de retroceso (←—) o mediante Control y U. Simplemente retrocede el cursor una posición y borra ese carácter, y una vez que haya sido suprimido, continúa tecleando la línea en cuestión.

Para suprimir completamente una línea que está justamente tecleándose en ese momento, pulsa simultáneamente Control y H.

Para corregir líneas de programa pertenecientes al programa que reside en ese momento en memoria, simplemente teclea la nueva línea que desees usando el mismo número de línea que la que quieres corregir. El BASIC automáticamente reemplazará la línea vieja con esta línea nueva.

Para suprimir todo el programa que reside en ese momento en memoria, teclea el comando NEW. El BASIC interpreta que debe 'renovar' todo el contenido de la memoria, porque quieres empezar con un programa nuevo.

3.11 TECLAS ESPECIALES

El BASIC admite diversas teclas especiales como las siguientes:

3.11.1 Teclas funcionales

Los ordenadores SV tienen 10 teclas funcionales pre-definidas en fábrica. La función presente de esas teclas se muestra en la última línea de la pantalla, y el usuario puede cambiar la definición a utilizar en un programa, mediante la instrucción KEY. Los valores iniciales son:

F1	color
F2	auto[10,10] |ENTER|
F3	goto
F4	list



F5	run |ENTER|
F6	color 15,4,5 |ENTER|
F7	cload"
F8	cont |ENTER|
F9	list. |ENTER|
F10	|CLS| run |ENTER|

Las teclas funcionales también se utilizan para establecer "cepos" y poder atrapar determinados eventos. Véase las instrucciones ON KEY GOSUB y KEY ON/OFF/STOP para más detalles.

3.11.2 Tecla de PARE y SIGA

Cuando el BASIC está en el modo directo, la tecla STOP no tiene ningún efecto ya que no se está realizando ninguna operación.

Cuando el BASIC está ejecutando el programa residente en memoria, si se pulsa la tecla STOP hace que se PARE temporalmente la ejecución del programa. El BASIC presenta la imagen del cursor para indicar que la ejecución ha sido suspendida. Al pulsar una segunda vez la tecla STOP, hace que SIGA la ejecución del programa en el punto en que se paró.

Sin embargo, si se pulsan simultáneamente las teclas Control y STOP, el BASIC termina la ejecución del programa y vuelve al modo directo con el siguiente mensaje:

```
Break in nnnnn
```

que indica que se ha producido una 'ruptura' precisamente cuando estaba ejecutando la línea de programa con número nnnnn.

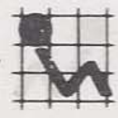
3.12 MENSAJES DE ERROR

Si por un error se provoca la terminación del programa que se estaba ejecutando, el BASIC muestra un mensaje de error. Para una lista completa de los códigos y mensajes de error en BASIC, consultese el Apéndice A.

3.13 ENTRADA Y SALIDA

3.13.1 Ficheros de datos

Un fichero es una colección de datos que guardan cierta relación entre sí, y que se conservan en otro medio de almacenamiento, distinto de la memoria de escritura y lectura. Puede ser por lo tanto, cassette o diskette.



Hay dos categorías de ficheros; i.e.: **secuenciales** y **directos** (también mal llamados aleatorios).

Con el fin de mantener ordenadamente esos ficheros, deben especificarse dos datos identificativos: el número del fichero y el nombre del fichero:

3.13.1.1 Número del fichero

El número del fichero es lo que el ordenador usa para referirse a un fichero dado, es un número **unívoco** asociado con cada fichero que se manipule en un programa, y asignado en el momento de la apertura del fichero. En realidad identifica el **cauce** que el ordenador usa para el envío de información desde la memoria al fichero y desde el fichero a la memoria.

3.13.1.2 Designando el fichero

Todo fichero depositado en un diskette o en un cassette, queda identificado por su nombre de fichero, que es un literal de la forma:

dispositivo: nombre-de-fichero

El nombre de la unidad de disco le indica al BASIC en cuál de las posibles ductoras conectadas al sistema, debe buscar el fichero mencionado; y el nombre de fichero le dice cuál es el fichero que requieres de todos los que haya grabados en el diskette o cassette alojado en esa unidad. Algunas veces, no es necesario especificar completamente estos datos. Por ejemplo, cuando lo que se quiere es recuperar del cassette el primer fichero que haya, podemos omitir tanto el número de fichero como el nombre.

El signo dos-puntos (:) es parte del nombre del **dispositivo**. Siempre que se estipula un determinado dispositivo, debes incluir el dos puntos, incluso aunque no menciones el nombre del fichero.

Recuerda encerrar entre comillas la especificación del fichero con el que vas a trabajar. Por ejemplo:

LOAD " dispositivo : nombre de fichero "



3.13.1.2.1 Nombre de dispositivo

Se pueden usar cuatro caracteres como máximo, excluyendo el dos puntos (:).

Nombres de dispositivos:

KYBD: Teclado (Keyboard) sólo para entrada. Para BASIC en cassette y diskette.

SCRN: Pantalla (Screen). Sólo para salida. Para BASIC en cassette y en diskette.

LPT: Impresora (Printer). Sólo para salida. Para BASIC en cassette y diskette.

CAS: Lectograbadora de cinta en cassette. Como equipo de almacenamiento externo. Entrada y salida. Para BASIC en cassette y diskette.

1: Primera lectograbadora de diskette. Entrada y salida. Para BASIC en cassette y diskette.

2: Segunda lectograbadora de diskette. Entrada y salida. Para BASIC en cassette y diskette.

3.13.1.2.2 Nombre del fichero

El nombre del fichero debe ser conforme con las siguientes reglas:

Para ficheros en cassette: el nombre debe constar de dos partes separadas por un punto (.)

nombre. extensión

El máximo número de caracteres para el nombre es seis y para la extensión es tres. Los caracteres adicionales serán truncados.

Si aparece un punto decimal en un nombre de fichero antes de que haya seis caracteres, el nombre se rellena de blancos hasta la sexta posición, y los siguientes tres caracteres posteriores al punto se consideran como la extensión.



Si el nombre del fichero tiene más de seis caracteres, el BASIC automáticamente colocará un punto decimal después del sexto carácter y usará los siguientes tres como la extensión del nombre del fichero.

3.13.2 Pantalla

El procesador de imágenes de video TMS9918/9929 admite imágenes con texto y gráficos en la pantalla. La pantalla permite al ordenador comunicarte el resultado de las acciones que le has mandado, mediante textos, caracteres especiales, motas, líneas o figuras más complejas en color o en blanco y negro.

Con la palabra TEXTO hacemos referencia a todos los caracteres del repertorio admitido: letras, cifras, signos y todos los caracteres especiales permitidos.

Puedes disfrutar de la capacidad de tu ordenador dibujando gráficos con los caracteres especiales para líneas y recuadros. Y no pierdas de vista los SPRITES. Puedes crear imágenes parpadeantes, invertidas, invisibles o resaltadas, con la ayuda de los comandos BASIC. En ambos modos de pantalla de alta y baja resolución, todos los puntos que componen la retícula pueden ser señalados. La pantalla pueden considerarla dividida en tres zonas o capas, una situada en la parte superior de la otra. Comenzando a partir de la más inferior, son el borde, el fondo y el plano frontal.

Se pueden mostrar un total de 16 colores, estando cada uno caracterizado por un número:

Número de color #	Tinte o color
0	Transparente
1	Negro
2	Verde medio
3	Verde pálido
4	Azul oscuro
5	Azul pálido
6	Rojo oscuro
7	Ciano
8	Rojo medio
9	Rojo pálido
10	Amarillo oscuro
11	Amarillo pálido
12	Verde oscuro
13	Magenta
14	Gris
15	Blanco



Aunque la apariencia de cada color puede variar dependiendo del aparato de TV o monitor que manejes. Ajustando el sintonizador de color puede ayudar a que los colores se correspondan exactamente con los mencionados.

3.13.2.1 Modo Texto

Es el modo prescrito para la imagen en pantalla cuando se pone en marcha el ordenador. Pero también puedes fijarlo mediante el comando SCREEN 0. En este modo comunicas tus deseos al ordenador mediante el teclado.

Con textos, el fondo cubre totalmente el borde de la pantalla, mientras que en el plano frontal aparecen todos los caracteres que expongas; i.e.: el texto.

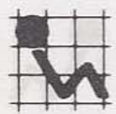
Los caracteres están dispuestos según 24 líneas horizontales, numeradas de 1 a 24, y de arriba hacia abajo. En cada línea puede haber 39, 40 u 80 caracteres. El valor prescrito para omisiones es 39 cuando no tienes cartucho de 80 columnas; o es 80 si has instalado ese cartucho. Con el comando WIDTH 40 puedes exponer en pantalla hasta 40 caracteres por cada línea. La numeración de las columnas comienza en 1 y va de izquierda a derecha. Los números que indican la posición de un carácter, línea y columna, se usan en los siguientes comandos o funciones:

LOCATE POS CSRLIN

La esquina superior izquierda de la pantalla tiene por tanto como coordenadas (1,1). Puedes usar la instrucción PRINT para situar cualquier carácter en cualquier posición de la pantalla. Si no mencionas explícitamente el sitio, el carácter aparecerá en la posición señalada por el cursor, y si lo que quieres es poner una cadena de caracteres, irán apareciendo de izquierda a derecha sobre la línea.

Cuando en su desplazamiento, el cursor alcance el final de la línea 24, la imagen se 'avanzará' hacia arriba una línea, de forma que la que era la línea 1 desaparece de la pantalla, permitiéndote que continúes exponiendo caracteres en la línea inferior.

La línea 24 sólo se utiliza habitualmente para mostrar los valores corrientes de las teclas funcionales. Sin embargo, es posible escribir encima de esa línea.



Los comandos o funciones aprovechables para mostrar información en el modo texto son:

CLS	POS	TAB
COLOR	PRINT	WIDTH
CSRLIN	SCREEN	WRITE
LOCATE	SPC	

3.13.2.2 Modo Gráficos

Hay dos modos gráficos con distinta resolución. Los comandos o funciones empleados para generar imágenes son los siguientes:

CIRCLE	LINE	PSET
COLOR	PAINT	PUT
DRAW	POINT	SCREEN
GET	PRESET	

Alta resolución

Se estipula mediante el comando SCREEN 1. Hay 256 motas horizontales (pixels) y 192 motas en el sentido vertical. Estas motas (pixels) se numeran de izquierda a derecha y de arriba a abajo, por lo que la esquina superior izquierda tiene como coordenadas (0,0). También pueden mostrarse en este modo gráfico los caracteres del repertorio, ocupando el mismo espacio que en el modo texto.

Baja resolución

Se estipula mediante el comando SCREEN 2. Hay 64 motas horizontales y 48 en el sentido vertical. La numeración es similar a la del modo de alta resolución.

3.13.3 Entrada/Salida

Cualquier clase de entrada/salida puede ser tratada como si fuera una entrada/salida hacia un fichero.

3.13.3.1 Sonido y música

Puedes crear sonido con tu ordenador mediante los comandos:

BEEP Emite un leve pitido.

SOUND Genera una nota sencilla con la frecuencia y duración deseadas.

PLAY Produce todo un chorro de notas para componer una melodía de acuerdo con unas determinadas reglas.



3.13.3.2 Juegos

Los 'joysticks' de palanca, bastón, bola, etc. son muy aprovechables en ambientes interactivos. El BASIC admite además de ellos, los tableros graficadores a base de coordenadas. Consulta para más detalles los siguientes comandos o funciones:

STICK   STRIG



COMANDOS, INSTRUCCIONES Y FUNCIONES DEL BASIC

4.1 COMANDOS, INSTRUCCIONES Y FUNCIONES EXCEPTO ENTRADA/SALIDA

4.1.1 Comandos excepto entrada/salida

Tratamos en este apartado de los comandos del BASIC, exceptuando los correspondientes a entrada y salida.

4.1.1.1 AUTO

Propósito: Generar automáticamente un número de línea después de pulsar |ENTER|.

Versiones: Cassette, diskette.

Formato: AUTO [[<nurolin>] [, [<salto>]]]

Observaciones: AUTO comienza a numerar en <nurolin> e incrementa cada número de línea subsecuente en <salto>. Lo prescrito para ambos valores es 10. Si <nurolin> está seguido de una coma pero no se menciona ningún <salto>, se adopta el último incremento especificado en un comando AUTO.

Si AUTO genera un número de línea que ya se está usando, se muestra un asterisco después de ese número de línea para avisar al usuario que lo teclee, sustituirá a la línea existente. Sin embargo, pulsando |ENTER| inmediatamente después del asterisco, conservará esa línea y generará el siguiente número de línea.

AUTO se termina tecleando CTRL-C o CTRL-STOP. La línea en que se teclea CTRL-C no se conserva. Después de pulsar CTRL-C, BASIC vuelve al nivel de comando.

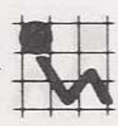
Ejemplo:

```
AUTO
```
Genera los números de línea 10, 20, 30, ...

```
AUTO 20, 5
```
Genera los números de línea 20, 25, 30, ...

```
AUTO 100,
```
Genera los números de línea 100, 105, 110, ...
El salto es 5 dado que el comando AUTO previo había estipulado que el salto fuera 5.

```
AUTO,3
```
Genera los números de línea 0, 3, 6, ...



4.1.1.2 CONT

Propósito: Hacer que continúe la ejecución del programa después de que se ha pulsado CTRL-C, o se haya ejecutado una instrucción STOP o una END.

Versión: Cassette, diskette.

Formato: CONT

Observaciones: La ejecución sigue en el punto en que se produjo el corte. Si el corte ocurrió después de una pregunta estipulada en una instrucción INPUT, la ejecución continúa volviendo a mostrar la pregunta, o el signo de interrogación pertinente.

CONT habitualmente se usa con STOP para depuración del programa. Cuando se para la ejecución, se pueden examinar los valores intermedios y se pueden cambiar usando comandos en modo directo. La ejecución puede luego reanudarse con el comando CONT, o con el comando GOTO mencionando un número de línea para que la ejecución siga a partir de él. Además, puede usarse para continuar la ejecución después de un error.

CONT no es válido si el programa se ha editado durante el corte. La ejecución no puede continuar si durante el corte se ha provocado un error en modo directo.

Ejemplo: Crear un bucle. Durante la ejecución del programa, se interrumpe pulsando simultáneamente |CTRL| y |STOP|.

```
10 FOR I = 1 TO 9
20 PRINT I;
30 NEXT
40 PRINT
50 GOTO 10
RUN
1 2 3 4 5 6 7 8 9
1 2 3 4 5 C
Break in 20
Ok
CONT
6 7 8 9
1 2 3 4 5 6 7 8 9
.
.
.
.
```

4.1.1.3 DELETE

Propósito: Suprimir líneas del programa.

Versiones: Cassette, diskette.

Formato: DELETE {[<nurolin1>] [- <nurolin2>]}

Observaciones: La primera línea a suprimir es la mencionada en <nurolin1>. La última línea a suprimir es la mencionada por <nurolin2> .

Puede usarse en lugar de los números y del guión un simple punto (.) para indicar la línea corriente. Si no se menciona ningún número de línea, se produce un error de "cita a función ilegal".

BASIC siempre vuelve al nivel de comandos después de que se ejecute un comando DELETE.

Ejemplo: Supongamos que hemos metido el siguiente programa:

```
10 FOR H = 0 TO 23
20 FOR M = 0 TO 59
30 FOR S = 0 TO 59
40 CLS
50 PRINT H ":" M ":" S
60 BEEP
70 FOR T = 1 TO 50
80 NEXT T
90 NEXT S
100 NEXT M
110 NEXT H
```

```
DELETE 10
```
Se suprime la línea 10

```
DELETE .
```
Se suprime la línea 110 (la última metida).

```
DELETE 60-80
```
Se suprimen las líneas 60, 70, 80.

```
DELETE -100
```
Se suprimen las líneas 20, 30, 40, 50, 90 y 100.



4.1.1.4 LIST

Propósito: Listar todo el programa, o parte de él.

Versiones: Cassette, diskette.

Formato: LIST [[<nurolin1>] [-[<nurolin2>]]]

Observaciones: Ambos <nurolin> deben estar en la gama 0 a 65529. La primera línea a listar es la mencionada en <nurolin1> , y es la única cuando no se menciona el segundo parámetro del comando. La última línea que se lista es la <nurolin2> .

Si se mencionan el guión y <nurolin2> únicamente, se listan todas las líneas desde el comienzo del programa hasta la línea mencionada por <nurolin2>.

Si se omiten ambos números de línea, se lista todo el programa.

Se puede usar simplemente un punto (.) para indicar el número de línea corriente.

El listado se interrumpe pulsando simultáneamente |CTRL| y |STOP|. El listado se suspende temporalmente usando |STOP|, y se reanuda pulsando |STOP| por segunda vez.



Ejemplo: Teclea primeramente el siguiente programa:

```
10 REM DEMOSTRACION PROGRAMA
20 DEF A - Z
30 J = 4: A = 16: B = 80: S = 8
40 ZZ = RND (- TIME)
50 SCREEN 1
60 FOR K = A TO B STEP S
70 C = INT (RND (1) *16)
80 LINE (K,K) - (255 - K, 191 - K),
   C, BF
90 C2 = INT (RND (1) *16)
100 IF C = C2 THEN 90 ELSE COLOR C2
110 FOR I = K TO 255 - K STEP J
120 LINE (I,K) - (255 - I, 191 - K)
130 NEXT
140 FOR I = 191 - K TO K STEP -J
150 LINE (K,I) - (255 - K, 191 - I)
160 NEXT
170 FOR Z = 1 TO 1000: NEXT: NEXT
180 FOR Z = 0 TO 1000: NEXT
190 SWAP A,B
200 S = -S
210 GOTO 50
```

LIST

Se lista en pantalla todo el programa.

LIST 10

Unicamente se lista la línea 10

LIST 10 - 30

Se listan de la línea 10 hasta la 30 inclusives.

LIST - 100

Se lista desde la primera línea, i.e., la de menor número de línea, hasta la línea 100.

LIST 130-

Se lista a partir de la línea 130 hasta el final del programa.



4.1.1.5 LLIST

Propósito: Listar por impresora todo el programa o parte de él.

Versiones: Cassette, diskette.

Formato: LLIST [[<nurolin1>] [- [<nurolin2>]]]

Observaciones: Consulta el comando LIST para mayor detalle.

Ejemplo: Consulta el comando LIST para mayor detalle.

4.1.1.6 NEW

Propósito:	Borrar todo el programa y los datos que haya en la memoria en ese momento.
Versiones:	Cassette, diskette.
Formato:	NEW
Observaciones:	Hace que se cierren todos los ficheros, y que se quite el modo de **rastreo** si estaba puesto.

NEW se usa habitualmente para dejar libre la memoria antes de meter un nuevo programa. BASIC vuelve siempre al nivel de comando después de haber ejecutado NEW.



4.1.1.7 RENUM

Propósito: Para cambiar la numeración de líneas de programa.

Versiones: Cassette, diskette

Formato: RENUM [[<nurolinnue>] [,[<nurolinvie>] [, <salto>]]]

Observaciones: <nurolinnue> señala el primer número de línea a usar en la nueva secuencia. Por omisión, se adopta 10.
<nurolinvie> es la línea en el programa existente a partir de la cual se cambia la numeración. Si se omite, se adopta la primera línea del programa.
<salto> es el incremento a usar en la nueva secuencia. Por omisión se toma 10.

El comando RENUM también cambia todos los números de línea que sirven de referencias en instrucciones como GOTO, GOSUB, THEN, ELSE, ON...GOTO, ON...GOSUB y ERL, asignándoles los nuevos números de línea resultantes. Si después de una de estas instrucciones apareciera un nuevo número de línea que no existe en el programa, aparecerá un mensaje indicando:

"Undefined line nnnnn in mmmmm"

para indicar que la línea no está definida. La referencia incorrecta a un número de línea (nnnnn) no se ve alterada por RENUM, pero sí se ha cambiado a un nuevo valor el número de la línea correspondiente (mmmmm).

Nota: No puede usarse RENUM para cambiar el orden de las líneas del programa (por ejemplo, RENUM 15,30 cuando el programa tiene tres líneas solamente, numeradas 10, 20, 30) o para crear números de línea mayores de 65529. En estos casos se producirá un error de 'cita de función ilegal'.



Ejemplo: RENUM

Se cambia la numeración de todo el programa, comenzando en 10 y saltando de 10 en 10.

RENUM 50, 40, 5

Renumera las líneas situadas por encima de la 40, con una nueva numeración comenzando en 50 y con un salto de 5.

RENUM , , 30

Renumera todas las líneas del programa con un incremento de 30, y comenzando la nueva numeración a partir de 10 (tomado por omisión).

RENUM ,5, 2

Renumera la línea 5 como 10, y a partir de ahí todas en saltos de 2.

RENUM 5, ,15

Numera la primera línea como 5, y todas las demás en saltos de 15.

RENUM 3, 5

Numera la línea 5 como 3, y a partir de ahí, todas de 10 en 10.

Mete el siguiente programa y los comandos respectivos. Lista el programa para observar los cambios:

```
   COLOR 15, 4
   SCREEN
15 LINE (50,50) - (205,141), 8
19 LINE (50,141) - (205,50), 8
23 CIRCLE (128,96), 90, 8
30 PAINT (135,125), 8
40 GOTO 40
```



4.1.1.8 RUN

Propósito: Ejecutar un programa residente ya en memoria, o traído de un equipo de almacenamiento externo.

Versiones: Cassette, diskette

Formato: RUN [<nurolin>]
RUN [<nomefich>] [, R]

Observaciones: Si se menciona <nurolin> , la ejecución comienza a partir de esa línea, en caso contrario, comienza en la línea con número inferior.

Con el segundo formato, el comando RUN carga un fichero del diskette o del cassette en la memoria y ejecuta el programa. RUN suprime el programa que hubiera en memoria, y cierra todos los ficheros antes de cargar el programa a ejecutar. Si se incluye la opción R, los ficheros de datos se mantienen abiertos.

Ejemplo: Teclea el siguiente programa y los comandos.

```
10  PRINT 10 "x";
20  PRINT 20 "=";
30  M = 10 * 20
40  PRINT M
RUN
10 x 20 = 200
Ok
RUN 30
200
Ok
```

El siguiente ejemplo hace que se cargue el programa "TEST" de la unidad de disco 1, y que se ejecute.

```
RUN "1: TEST"
```

Si el programa está grabado en cassette, teclea el siguiente comando:

```
RUN "TEST"
```




4.1.1.9 TRON/TROFF

Propósito: Rastrear la ejecución del programa.

Versiones: Cassette, diskette.

Formato: TRON/TROFF

Observaciones: Como ayuda en la depuración de los programas, se puede dar el comando TRON (o incluirlo como una instrucción en el programa) para que ponga en marcha el mecanismo que muestra en pantalla cada número de línea del programa a medida que se está ejecutando. Los números aparecen encerrados entre corchetes. El rastreo se quita mediante el comando TROFF, o con el comando NEW.

Ejemplo:

```
10 CLS
20 LOCATE 10, 5
30 PRINT "TEST"
RUN
```

Se limpia la pantalla y luego se muestra lo siguiente:

```
[10]
                    [20] TEST
Ok
```



4.1.1.10 CLEAR

Propósito: Anular todas las variables (las numéricas a cero y las litéricas a nulo), cerrar todos los ficheros abiertos, y -opcionalmente- repartir el espacio de memoria.

Versiones: Cassette, diskette.

Formato: CLEAR [, [<exprnum1>] [, <exprnum2>]]

Observaciones: CLEAR deja libre toda la memoria usada para datos sin borrar el programa existente en memoria. Además, las tablas quedan como no definidas, y cualquier información estipulada mediante instrucciones DEF queda sin efecto.

<exprnum1> fija la máxima dirección de memoria disponible para su uso por BASIC.

<exprnum2> estipula el espacio reservado para **stack** (donde apilar las direcciones de retorno de subrutinas).

Nota: En versiones anteriores de BASIC, <exprnum1> fijaba la cantidad de espacio para constantes literales; y <exprnum2> el tope de memoria.

Ejemplo:

CLEAR

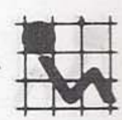
Anula todos los datos de la memoria sin borrar el programa.

CLEAR 32768

Anula los datos y estipula que el tope de memoria está en la dirección 32768.

CLEAR 32768, 1000

Borra los datos; establece como tope de memoria utilizable la dirección 32768; y reserva 1000 **octetos** como espacio de memoria para ser usado como **stack**.



4.1.1.11 DATA

Propósito: Conservar las constantes numerales y literales que van a ser sucesivamente apuntadas como valores de las variables mencionadas en las instrucciones READ que haya en el programa.

Versiones: Cassette, diskette.

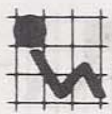
Formato: DATA <lista de constantes>

Observaciones: La <lista de constantes> puede contener constantes numéricas de cualquier clase; i.e., coma flotante, coma fija, o enteros (pero no expresiones numéricas). Las constantes literales incluidas en las instrucciones DATA, deben estar encerradas entre comillas únicamente cuando contienen comas, dos puntos o sus blancos delanteros o traseros son significativos. De lo contrario, no necesitan -aunque obviamente pueden- ser encerradas entre comillas.

Las instrucciones DATA no son ejecutables y pueden colocarse en cualquier parte del programa. Pueden contener tantas constantes como se admitan en una línea de programa (separando las constantes por comas); y se pueden usar en un programa cualquier número de instrucciones DATA que se desee.

Estas constantes van siendo sucesiva y correlativamente asignadas a las variables mencionadas en las instrucciones READ,. en el orden en que se van ejecutando dichas instrucciones. Por lo tanto, pueden considerarse como una lista secuencial de elementos, independientemente de cuántos elementos haya en una línea o de cuántas líneas haya en un programa.

La clase de variable (numeral o literal) de las instrucciones READ debe concordar exactamente con la constante correspondiente de la instrucción DATA. Se puede hacer que BASIC repunte sus indicadores internos hacia el primer elemento de una lista DATA predeterminada, mencionando en la instrucción RESTORE el número de línea de dicha lista.

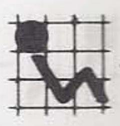


Ejemplos:

```
10 FOR I = 1 TO 3
20 READ NOME$(I), EDAD%(I)
30 NEXT
40 DATA JULIO, 42, JULIA, 24, JULIAN, 21
```

En este programa hay una lista de 6 constantes, alternativamente literales y numerales, en la línea numerada 40. Si quisieramos que hubiera el signo dos puntos detrás de los nombres, tendríamos que cambiar la línea 40 para que fuera:

```
40 DATA "JULIO:", 42, "JULIA:", 24,
   "JULIAN:", 21
```

4.1.1.12 DIM

Propósito: Ocupar el espacio correspondiente a una **tabla** y definir el límite superior de cada uno de los subíndices que identifica los elementos de la tabla.

Versiones: Cassette, diskette.

Formato: DIM {<nomevar> (<sub1>,<sub2>, ...)} [, ...

Observaciones: Si se usa una variable perteneciente a una tabla, mediante un nombre y un uno o varios subíndices, sin una instrucción DIM previa, el máximo valor de los subíndices se adopta como 10. Si se usa un subíndice mayor que el máximo mencionado en la instrucción DIM, se produce un error de 'subíndice fuera de gama'. El valor mínimo para un subíndice es siempre cero. El número máximo de subíndices -dimensiones- de una tabla es 255. El número máximo de elementos por cada una de las dimensiones es 32767. Pero sin embargo, ambos números están limitados por el tamaño de la memoria y la longitud de la instrucción.

Una tabla sólo puede dimensionarse una vez. En caso que sea necesario cambiar sus dimensiones, se usa la instrucción ERASE para 'desocupar' el espacio correspondiente y poder darle nuevas dimensiones.

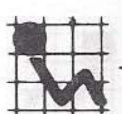
Ejemplo: El siguiente ejemplo crea dos tablas. Una tabla literal monodimensional llamada M$ y con 13 elementos, designados M$(0) a M$(12); y una tabla numérica real llamada D, con 13 elementos de nombres D(0) a D(12).

```
10 DIM M$(12)
20 DIM D(12)
30 PRINT "AÑO" TAB(5) 1984
40 FOR I = 1 TO 12
50 READ M$(I), D(I)
60 PRINT M$(I) TAB(6) D(I)
70 NEXT
80 DATA ENE, 31, FEB, 29, MAR, 31
   ABR, 30, MAY, 31, JUN, 30, JUL, 31,
   AGO, 31, SEP, 30, OCT, 31, NOV, 30,
   DIC, 31
RUN
AÑO 1984
ENE   31
FEB   29
MAR   31
ABR   30
MAY   31
JUN   30
JUL   31
AGO   31
SEP   30
OCT   31
NOV   30
DIC   31
Ok
```

Ahora cambia el programa para que sea:

```
10 DIM A $ (12, 1)
20 FOR J = 0 TO 12
30 FOR K = 0 TO 1
40 READ A$ (J,K)
50 PRINT A$ (J,K)
60 NEXT K
70 PRINT
80 NEXT J
90 DATA AÑO, 1984, ENE, 31, FEB, 29,
   MAR, 30, ABR, 30, MAY 31, JUN, 30,
   JUL, 31, AGO, 31, SEP, 30, OCT, 31,
   NOV, 30, DIC, 31
```

En este ejemplo, se crea una tabla literal bidimensional A$, con 26 elementos, desde A$(0,0) hasta A$(12,1).



4.1.1.13 DEFINT
DEFSNG
DEFDBL
DEFSTR

Propósito: Designar la clase de variable de acuerdo con la primera letra de su nombre.

Versiones: Cassette, diskette.

Formato:

```
DEFINT  {<gama de letras>}, ...
DEFSNG  {<gama de letras>}, ...
DEFDBL  {<gama de letras>}, ...
DEFSTR  {<gama de letras>}, ...
```

Observaciones: Las instrucciones DEFINT|SNG|DBL|STR especifican que las variables que empiecen con una letra de la <gama de letras> , sean respectivamente numerales enteras, numerales en simple precisión, numerales en doble precisión y literales. Sin embargo, un carácter trasero designador de clase de variable (%, !, #, $) siempre tiene prioridad sobre una instrucción DEFxxx.

(Véase el final de la sección 3.6 para más detalles sobre los caracteres designadores de las clases de variables).



Ejemplo:

```
10  DEFINT A, B
20  DEFSNG E, F
30  DEFDBL D, G-I
40  DEFSTR C
50  AVERAGE = (2+3+6)/3 : PRINT PROMEDIO
60  EVASAVG = (1+3+1)/3 : PRINT EVASAVG
70  DANSAVG = (2+4+1)/3 : PRINT DANSAVG
80  COMMENT = "PROMEDIO ES BAJO" : PRINT
    COMMENT
RUN
3
1.66667
2.3333333333333
PROMEDIO ES BAJO
Ok
```

La línea 10 especifica que todas las variables que comiencen con la letra A o con la letra B sean variables numerales enteras.

La línea 20 hace que todas las variables que comiencen con la letra E o la letra F sean variables numéricas en simple precisión.

La línea 30 determina que todas las variables que comienzan con las letras D, G, H, I sean variables numéricas de doble precisión.

La línea 40 especifica que todas las variables que comienzan con la letra C sean consideradas variables literales.



4.1.1.14 DEF FN

Propósito: Para definir y designar una función escrita por el usuario.

Versiones: Cassette, diskette.

Formato: DEF FN <nomefunc> [(lista de argumentos)] = <fórmula a aplicar>

Observaciones: El <nomefunc> debe ser un nombre de variable legal. Ese nombre, precedido por FN, se convierte en el nombre de la función. La <lista de argumentos> consta de aquellos nombres de variables que intervienen en la <fórmula a aplicar> para calcular el resultado de la función, y son sustituidos por los argumentos **actuales** cuando se cita la función. Los argumentos de la lista se separan mediante comas.

La <fórmula a aplicar> es una expresión válida en BASIC que utiliza variables, constantes, y funciones del BASIC, combinadas por los operadores del BASIC. Está limitada a una sola línea.

Los nombres de los argumentos que aparecen en la definición de la función solamente sirven como parámetros **formales;** pero no afectan a las posibles variables que haya en el programa con ese mismo nombre.

En la expresión que define la función puede aparecer cualquier nombre de variable, aunque no se mencione en la lista de argumentos. Si así es, el valor actual que tenga esa variable cuando se cita la función es el que se toma para evaluar la fórmula que define la función.

Las variables mencionadas en la lista de argumentos representan, de forma biunívoca, los argumentos **actuales** -valores a utilizar realmente- que deben ser mencionados al citar la función.

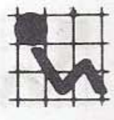
Si se designa una cierta clase de función, mediante un carácter designador, el valor de la fórmula a aplicar queda obligado a ser de la misma clase que se ha especificado en el nombre de la función. Si no correspondiera el resultado de la evaluación con la clase de función especificada, o el valor actual del argumento no correspondiera con el valor **formal** del argumento mencionado en la especificación de la función, se produciría un error de 'discordancia de clases'.

La instrucción DEF FN que especifica una determinada función, debe ser ejecutada antes de que pueda citarse dicha función. Si se hiciera lo contrario, se produciría un error de 'función de usuario indefinida'. También se puede usar DEF FN como comando, en el modo directo.

Ejemplo:

```
10 DEF FNAREA (B,H) = B * H/2
20 INPUT "BASE ="; BASE
30 INPUT "ALTURA ="; ALTURA
40 PRINT "AREA ES" FNAREA (ALTURA, BASE)
RUN
BASE = ? 3
ALTURA = ? 6
AREA ES 9
Ok
```

La línea 10 especifica la función FNAREA como el semiproducto de los argumentos B y H. Cuando se cita la función, los argumentos **actuales** son base y altura, tal y como se refleja en la línea 40.



4.1.1.15 DEFUSR

Propósito: Especificar la dirección de comienzo de una subrutina en código máquina, que puede ser citada posteriormente mediante la función USR.

Versiones: Cassette, diskette.

Formato: DEFUSR [<dígito>] = <exprnum entera>

Observaciones: <dígito> puede ser cualquier cifra entre 0 a 9. Por lo tanto, pueden especificarse hasta 10 rutinas diferentes. Si se omite el 'dígito' se adopta DEFUSR0.

El valor de <exprnum entera> es la dirección en que comienza la subrutina en código máquina.

En un programa, puede aparecer cualquier cantidad de instrucciones DEFUSRn, que van redefiniendo las anteriormente especificadas y que posean el mismo dígito; de esta manera se permite citar tantas subrutinas en código máquina como sea necesario.

Ejemplo:

```
100 DEF USR0 = 24000
200 X = USR0 (Y * 5)
```

Este ejemplo pone en acción una rutina que comienza en una celdilla de memoria cuya dirección absoluta es 24000.

4.1.1.16 ERASE

Propósito: Desocupar el espacio de una tabla.

Versiones: Cassette, diskette.

Formato: ERASE {<nombre de tabla> } [, ...

Observaciones: Se puede volver a dimensionar una tabla después de ejecutar la instrucción ERASE; o usar el espacio que ha quedado libre en la memoria para cualquier otro propósito.

Si se intenta cambiar las dimensiones de una tabla, sin haber primeramente 'desocupado' el espacio pertinente, se produce un error de 'redimensionamiento de tabla'.

Ejemplo:

```
10 PRINT FRE(0);
20 DIM A (50, 50)
30 PRINT FRE (0);
40 ERASE A
50 DIM A (10, 10)
60 PRINT FRE (0)
RUN

29126 8308 28148
Ok
```

Este ejemplo usa la función FRE, que proporciona la memoria disponible para ilustrar cómo puede usarse la instrucción ERASE. Si la tabla tiene de dimensiones (50, 50), ocupa 20K de memoria (de 29126 pasamos a 8308). Si sus dimensiones fueran (0, 10), se requeriría sólo 1K (pasamos de 29126 a 28148).

Teclea

```
DELETE 40
```

Y ejecuta el programa para ver lo que sucede.



4.1.1.17 END

Propósito: Para terminar la ejecución del programa, cerrando todos los ficheros y volviendo al nivel de comando.

Versiones: Cassette, diskette.

Formato: END

Observaciones: La instrucción END puede colocarse en cualquier parte del programa para terminar la ejecución del mismo. A diferencia de la instrucción STOP, la instrucción END no provoca la aparición del mensaje

BREAK ON nnnnn

No es imprescindible poner una instrucción END al final de un programa.

Ejemplo:

```
10  READ X
20  PRINT X
30  IF X > 100 THEN END ELSE GOTO 10
40  DATA 50, 200
RUN
50
200
Ok
```

A causa de la línea 30, el programa terminará si el valor de X sobrepasa de 100.



4.1.1.18 ERROR

Propósito: Provocar una situación de error (lo que permite simular uno ya definido, o que el usuario defina nuevos códigos de error).

Versiones: Cassette, diskette.

Formato: ERROR <exprnum entera>

Observaciones: El valor de <exprnum entera> debe estar en la gama de 0 a 255, y será el código de error con que se dote a la variable del sistema ERR, en el momento de ejecutar el comando. (El número de línea en que se provoca el error, queda reflejado en la variable del sistema ERL).

Si el valor mencionado en el comando ERROR ya está siendo usado por BASIC, el efecto es el mismo que si se hubiera producido realmente el error con ese código, y -normalmente- se mostraría en pantalla el correspondiente mensaje de error. Para definir tus propios códigos y mensajes de error, usa en el comando ERROR un valor superior a cualquiera de los usados por BASIC. (Véase Apéndice A con la lista de los códigos y mensajes de error; y es preferible que uses los valores más altos de manera que siga habiendo compatibilidad cuando se añadan más en las sucesivas versiones de BASIC). Este código de error definido por el usuario, puede luego ser manipulado convenientemente en una rutina de tratamiento de errores.

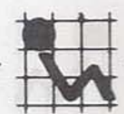
Si en una instrucción ERROR se menciona un código para el que no está definido ningún mensaje de error, BASIC responde genéricamente con el mensaje de "error no mostrable". La ejecución de una instrucción ERROR cuando no hay especificada ninguan rutina de tratamiento de errores, provoca la aparición del mensaje citado y que se detenga la ejecución del programa.

Ejemplo: En modo directo:

```
ERROR 10
Undefined array
Ok
```

O definiendo un código nuevo de error, pero sin asociarle mensaje

```
10 READ A$
20 IF A$ = "FALSO" THEN ERROR 250
30 DATA FALSO
RUN
Unprintable error in 20
Ok
```



4.1.1.19 FOR...NEXT

Propósito: Señalar el inicio de un bucle, i.e. la ejecución repetida de una determinada serie de instrucciones, un cierto número de veces.

Versiones: Cassette, diskette.

Formato: FOR <control> = <inicial> TO <ultimal> [STEP <salto>]

Observaciones: <Control> puede ser una variable entera, una variable en simple precisión o en doble precisión, mientras que <inicial> , <ultimal> , y <salto> son expresiones numéricas.

La variable de control del bucle, actúa como un contador cuyo valor se fija primeramente a <inicial> . Las líneas de programa que siguen a la instrucción FOR se ejecutan una vez hasta que se encuentra la instrucción NEXT, que marca el final del bucle. En ese momento se incrementa el contador en la cantidad especificada por <salto> .

Se efectúa una comprobación entre el valor resultante del contador y el valor de <ultimal> . Si no es mayor, el BASIC salta de nuevo a la instrucción siguiente a la instrucción FOR y vuelve a ejecutar otra vez todas las instrucciones hasta encontrar la instrucción NEXT. Si es mayor, la ejecución del programa continúa con la instrucción que sigue a la instrucción NEXT.

Si no se menciona la cláusula STEP, se supone que el salto a realizar en cada ronda es de 1. Si el valor de <salto> es negativo, el valor de <ultimal> ha de ser inferior de <inicial> , ya que se decrementará el contador a cada ronda que se efectúe. Y en este caso se ejecutará el bucle hasta que la variable de control alcance un valor inferior al de <ultimal> .

La serie de instrucciones que componen el bucle, se ejecutarán como mínimo una vez siempre que el valor de <inicial> multiplicado por el signo de <salto> sea menor que el valor de <ultimal> multiplicado por el signo de <salto> .

Los bucles FOR...NEXT pueden ser anidados; es decir, puede colocarse un bucle FOR...NEXT dentro completamente de otro bucle FOR...NEXT. Cuando se anidan bucles, cada uno debe tener un nombre diferente para la variable que controla el bucle como un contador. La instrucción NEXT que señala el final del bucle más interno, debe aparecer antes de la que señala el final del bucle más externo. Si los bucles anidados tienen el mismo punto final, se puede usar una sola instrucción NEXT para todos ellos.

El anidamiento de bucles FOR...NEXT está limitado únicamente por la memoria disponible.

La variable o variables de la instrucción NEXT pueden omitirse, en cuyo caso, la instrucción NEXT se hará corresponder con la instrucción FOR más reciente. Si se encuentra en el programa una instrucción NEXT antes de su correspondiente instrucción FOR, se emitirá un mensaje de error "NEXT without FOR" y se terminará la ejecución.

```
10 SUMA = 0
20 FOR X = 1 TO 100
30 SUMA = SUMA + X
40 NEXT
50 PRINT "LA SUMA DE ENTEROS DESDE
1 HASTA 100 es" SUMA
RUN
LA SUMA DE ENTEROS DESDE 1 HASTA
100 ES 5050
```

En el bucle del ejemplo se ejecutarán cien rondas, cada una de una instrucción (la 30) y la X variará de 1 a 100 en saltos de 1.

```
10 SCREEN 2
20 FOR X = 0 TO 20 STEP 2
30 FOR Y = 20 TO 0 STEP -2
40 PSET (X, Y)
50 NEXT Y
60 NEXT X
```

En este segundo ejemplo, hay dos bucles anidados. En el más externo el salto supone un incremento de 2 a cada ronda, y en el más interno un decremento también de 2. Por lo tanto, se ejecutarán 11 veces las instrucciones del bucle interno (la 40), por cada una de las 11 rondas efectuadas en el bucle externo.



4.1.1.20 GOSUB...RETURN

Propósito: Producir un **desvío** hasta un número de línea, manteniendo constancia internamente del punto en que se produce, para poder **volver** a él al encontrarse con una instrucción RETURN.

Versiones: Cassette, diskette.

Formato: GOSUB <nurolin> RETURN

Observaciones: <nurolin> es CUALQUIER línea del programa, y puede ser mencionada tantas veces como se desee, y BASIC continuará su ejecución a partir de ese número de línea hasta que encuentre una instrucción RETURN.

Normalmente, <nurolin> corresponde al primer número de línea de una **subrutina** (i.e. serie consecutiva de instrucciones que ejerce una determinada tarea y cuyo final está marcado por una instrucción RETURN).

Las instrucciones GOSUB pueden colocarse en cualquier parte del programa, incluso dentro de otra o de la misma subrutina. Una subrutina puede contener más de una instrucción RETURN, si la lógica del programa lo exige. Las subrutinas pueden aparecer en cualquier parte del programa, pero se recomienda que sea fácilmente distinguible de la rutina principal.

Para prevenir la entrada inadvertida en una subrutina, puede precederse con instrucciones STOP, END o GOTO, para dirigir al programa de manera que esquive la subrutina. De lo contrario, se emitirá el mensaje de error 'RETURN without GOSUB' y se termina la ejecución.



Ejemplo:

```
10  INPUT "DIME DIA DE LA SEMANA" ; A
20  GOSUB 50
30  INPUT "MEMO"; M $
40  GOTO 10
50  IF A=1 THEN PRINT "LUNES"
60  IF A=2 THEN PRINT "MARTES"
70  IF A=3 THEN PRINT "MIERCOLES"
80  IF A=4 THEN PRINT "JUEVES"
90  IF A=5 THEN PRINT "VIERNES"
100 IF A=6 THEN PRINT "SABADO" ELSE
    PRINT "DOMINGO"
110 RETURN
```

Muestra cómo funciona una subrutina. El GOSUB de la línea 20 desvía el programa hacia la subrutina que comienza en la línea 50, y comienza a ejecutar las instrucciones siguientes hasta que se encuentra con la instrucción RETURN de la línea 110. En ese momento, vuelve atrás a la instrucción inmediatamente detrás de la que produjo el desvío, i.e. línea 30. La instrucción GOTO de la línea 40 impide que se vuelva a efectuar la subrutina por segunda vez.



4.1.1.21 GOTO

Propósito: Saltar incondicionalmente fuera de la secuencia normal del programa hasta un número de línea determinado.

Versiones: Cassette, diskette.

Formato: GOTO <nurolin>

Observaciones: Si <nurolin> es una instrucción ejecutable, esa instrucción y las siguientes serán ejecutadas. Si no lo es, la ejecución prosigue en la primera instrucción ejecutable encontrada después de ese número de línea.

Ejemplo:

```
100 PRINT "QUIERES OTRO JUEGO (S/N)"
200 A$ = INKEY$
300 IF A$ <> CHR$(83) AND A$ <> CHR$(78)
    THEN GOTO 200
400 IF A$ = CHR$(83) THEN GOTO 10
500 END
```

Muestra cómo se hace una bifurcación en un programa. La línea 300 hace primeramente una comprobación, y luego salta a la línea 200 si el resultado ha sido cierto. Igualmente ocurre con la línea 400.



4.1.1.22 IF...THEN...ELSE IF...GOTO...ELSE

Propósito: Elegir entre dos alternativas basándose en el resultado de una expresión de relación.

Versiones: Cassette, diskette.

Formato: IF <expresión> THEN {<serie1 de comandos> | <nurolin1>}
[ELSE {<serie2 de comandos> | <nurolin2>}]

Observaciones: Habitualmente, <expresión> es una combinación de operaciones lógicas y de relación que estipulan una **condición** cuya certeza o falsedad se comprueba en el momento de ejecutar la instrucción; pero puede también ser una expresión numeral entera de cuyo resultado sólo se considera el valor 0 para indicar falso, y el valor distinto de cero para indicar cierto.

Si el resultado es cierto, se ejecuta la serie de comandos que forman la cláusula THEN, o se produce un salto al número de línea mencionado en esa cláusula.

Si el resultado de la expresión es falso (0), se ignora la cláusula THEN, y se ejecuta en su lugar la serie de comandos o el salto mencionado implícitamente, cuando dicha cláusula está presente. Si no lo está, la ejecución continúa con la siguiente instrucción ejecutable.

También las alternativas IF...THEN...ELSE pueden anidarse, hasta un nivel limitado únicamente por la longitud de la línea de instrucciones. Si las instrucciones no contienen el mismo número de palabras ELSE y THEN, cada cláusula encabezada por ELSE se empareja con la cláusula THEN más cercana y anterior.

```
IF A=B THEN IF B=C THEN PRINT "A=C"
ELSE PRINT "A< >C"
```

El ordenador no expondrá "A< >C" cuando sea A < >B; sino que expondrá "A < >C" cuando sea A=B y también B< >C.

Si en el modo directo se da un comando IF...THEN con un número de línea, se producirá un error de 'línea no definida' a no ser que previamente se haya metido en el modo indirecto una línea de programa con dicho número de línea.



Ejemplo: IF 2 + 2 > 2 THEN PRINT "2 + 2 ES MAYOR QUE 2"
2 + 2 ES MAYOR QUE 2
Ok

Dado que 2 + 2< >2 es una condición cierta, se ejecuta la cláusula THEN.

IF 2=3 THEN PRINT "2 ES IGUAL QUE 3" ELSE PRINT "2=3 ES FALSO"
2=3 ES FALSO
Ok

Dado que 2=3 es falso, se ejecuta la cláusula ELSE.

```
10  PRINT "CARA O CRUZ (0 O 1)"
20  FOR I = 1 TO 5
30  PRINT I "ES";
40  INPUT N
50  IF N< >0 AND N< >1 GOTO 40
60  T = T + N
70  NEXT
80  PRINT "DE 5 TIRADAS, HAN SIDO
    "5 - T %CARAS Y " T "CRUCES"
RUN
CARA O CRUZ (0 O 1)
1 ES? P
? REDO FROM START
? 0
2 ES? 1
3 ES? 1
4 ES? 1
5 ES? 0
DE 5 TIRADAS, HAN SIDO 2 CARAS Y 3
CRUCES
Ok
```

La línea 50 comprueba si se ha pulsado 0 ó 1. Y si no ha sido ninguno de los dos, el programa se dirige a la línea 40.



4.1.1.23 INPUT

Propósito: Imponer valores a las variables durante la ejecución de un programa y mediante el teclado.

Versiones: Cassette, diskette.

Formato: INPUT [;] [" <pregunta> " {;|, }] <lista de variables>

Observaciones: Cuando se encuentra la instrucción INPUT, se hace una pausa en la ejecución del programa y se expone en pantalla un signo de interrogación para indicar que el programa está esperando que se le tecleen determinados valores. Si se incluye en la instrucción la <pregunta> , que ha de ser una constante literal, aparecerá en pantalla delante del signo de interrogación.

El programa espera hasta que se le vayan tecleando valores, que irá imponiendo sucesivamente en la variable o variables mencionadas en <lista de variables> . El número de valores tecleados debe concordar con el número de variables de la lista. Se pueden separar unos de otros mediante comas, o pulsando |ENTER|.

Las variables de la lista pueden ser numerales o literales según el nombre que posean, y están admitidas también las variables que son elementos de tablas. La clase de valor que se teclee, ha de ser homogénea con la variable correspondiente. (Los valores literales correspondientes a una variable literal de una instrucción INPUT no necesitan estar entrecomillados).

Respondiendo por teclado a la instrucción INPUT con un valor en desacuerdo con la variable correspondiente (literal en lugar de numeral, expresión en lugar de constante, etc.) provoca que aparezca en pantalla el mensaje '?Redo from start' para indicar que debe volverse a hacer desde el principio; y no se hace ninguna imposición de valor en ninguna variable hasta que se dé una respuesta válida.



Ejemplo:

```
10  INPUT "A y B";A,B
0 PRINT A+B
Ok
run
A y B? 10,  0
?Redo from start
A y B? 10,20
30
Ok
```

Respondiendo a una instrucción INPUT con demasiados valores, se provoca la aparición del mensaje "?Extra ignored" para indicar que se ignoran los valores extras, pero no los anteriores que se imponen a las variables, y después se pasa a ejecutar la siguiente instrucción.

```
run
A y B? 10,20,40
?Extra ignored
30
Ok
```

Puedes 'escaparte' de una instrucción INPUT tecleando CTRL y C, o bien simultáneamente CTRL y STOP. El sistema vuelve al nivel de comando y presenta la divisa típica "Ok". Tecleando el comando CONT la ejecución sigue a la instrucción INPUT.



4.1.1.24 LINE INPUT

Propósito: Imponer toda una línea (de hasta 254 caracteres) como valor de una variable literal, sin el uso de **delimitadores.**

Versiones: Cassette, diskette.

Formato: LINE INPUT [" <pregunta> ";] <varlit>

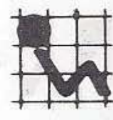
Observaciones: La <pregunta> es una constante literal que aparece en pantalla como aviso para la introducción del dato. No aparecerá ningún signo de interrogación a no ser que forme parte del literal <pregunta> . Todo lo tecleado hasta que se pulsa la tecla |ENTER| queda asignado a <varlit> .

Puedes escapar de la instrucción LINE INPUT tecleando CTRL y C o simultáneamente CTRL y STOP. BASIC regresa al nivel de comando y presenta el "Ok". Tecleando CONT se prosigue la ejecución en esa misma instrucción LINE INPUT.

Ejemplo:

```
LINE INPUT "DIRECCION:",N$
DIRECCION: URB. LAS HUERTAS Bq 9;
Piso 4,B
Ok
```

El ordenador te avisa que debes teclear un valor literal después de la pregunta "DIRECCION:". A no ser que pulses ENTER, no volverá a mostrarse la divisa "Ok".



4.1.1.25 LET

Propósito: Asignar el valor de una expresión a una variable.

Versiones: Cassette, diskette.

Formato: LET <variable> = <expresión>

Observaciones: Observa que la palabra LET es optativa; el signo igual es suficiente cuando se asigna el resultado de una expresión a una variable.

Ejemplo:

```
10 LET DI=3
20 LET D2=4
30 LET D3=5
40 SUMA = D1+D2+D3
50 PRINT SUMA
RUN
12
Ok
```

Ensaya el programa anterior suprimiendo la palabra LET de las líneas 10 a 30.



4.1.1.26 LPRINT LPRINT USING

Propósito: Exponer datos por la impresora.

Versiones: Cassette, diskette.

Formato: LPRINT [USING <conformatriz> ;]
[<lista de datos>]

Observaciones: La <lista de datos> es una serie de expresiones numerales y/o literales, que debieran estar separadas entre sí por comas o puntos y coma.

<conformatriz> es una variable o una constante literal que identifica y conforma la apariencia de los datos en la pantalla. (Se suele llamar también **horma de edición**).

La instrucción LPRINT presupone una impresora con anchura de 132 caracteres; por lo que BASIC inserta automáticamente un carácter de 'retorno de carro' o 'avance de línea' después de sacar 132 caracteres. Esto dará como resultado que se salte una línea de más cuando se impriman 132 caracteres, a no ser que la instrucción LPRINT termine con un punto y coma.

Ejemplo: Consulta las instrucciones PRINT y PRINT USING.



4.1.1.27 MID$

Propósito: Sustituir parte de un dato literal con otro.

Versiones: Cassette, diskette.

Formato: MID$ (<exprlit1> , <exprnum1> [, <exprnum2>]) = <exprlit2>

Observaciones: Los caracteres de <exprlit1> situados a partir de la posición <exprnum1> , son sustituidos por los caracteres de <exprlit2> ; y se sustituyen tantos como indique el valor de <exprnum2> .

Si se omite <exprnum2> se sustituyen hasta que se terminen los caracteres de una u otra expresión literal. Ninguna sustitución de caracteres efectuada con la instrucción MID$ puede nunca sobrepasar la longitud en caracteres de la expresión literal original.

Ejemplo:

```
10 A$="SCAN"
20 MID$(A$,2,3)="TAR"
30 PRINT A$
RUN
STAR
Ok
```

En la línea 20, el segundo, tercero y cuarto carácter de A$ son sustituidos por los tres caracteres del literal "TAR".

4.1.1.28 ON ERROR GOTO

Propósito: Designar un número de línea del programa a la que saltará en cuanto se produzca cualquier error durante la ejecución.

Versiones: Cassette, diskette.

Formato: ON ERROR GOTO <nurolin>

Observaciones: Se suele decir que la instrucción ON ERROR perpara un **cepo** para que al atrapar un error el programa salte a la línea dada por <nurolin> ; donde estará situada la rutina de tratamiento de errores.

Si no existe dicha línea en el programa, dará como resultado un error de 'número de línea sin definir'.

Para **desarmar** en cualquier momento el cepo que atrapa los errores, de manera que los errores que se produzcan posteriormente detengan la ejecución y emitan el mensaje de error pertinente, puedes ejecutar una instrucción ON ERROR GOTO 0.

Se recomienda que en todas las rutinas de tratamiento de errores, se ejecute una instrucción ON ERROR GOTO 0, para todos aquellos posibles errores para los que no se especifica la acción recuperadora.

Si ocurre un error durante la ejecución de la propia subrutina de tratamiento de errores, el BASIC procede en su forma habitual emitiendo el mensaje de error y terminando la ejecución. Lógicamente, si ha entrado en acción la rutina de tratamiento de errores es porque se ha **disparado** el cepo, con lo que no puede volver a intervenir durante la ejecución de esa rutina.



Ejemplo:

```
10  REM ADIVINAME UN NUMERO DEL
    1 AL 100
20  ON ERROR GOTO 80
30  A=1 + INT(100* RND (-TIME))
40  INPUT "DIME CUAL HE ELEGIDO";B
50  IF A >B THEN ERROR 80 ELSE IF A < B
    THEN ERROR 81
60  PRINT "ACERTASTE"
70  END
80  IF ERR = 80 THEN PRINT "TE PASAS"
    ELSE PRINT "NO LLEGAS"
85  ON ERROR GOTO 0
90  RESUME 40
100 END
```

Este programa es un juego de adivinanza de números. El 'cepo' para atrapar errores, se prepara en la línea 20; y se dispara con el código de error 80 o con el código de error 81 -que BASIC no usa- dependiendo de las comparaciones efectuadas en la línea 50. O no se dispara, pasando entonces a la línea 60 en que termina el juego.

La línea 85 se preocupa de los otros posibles errores que se puedan cometer.



4.1.1.29 ON...GOTO ON...GOSUB

Propósito: Saltar o desviarse hacia una de varias líneas posibles, según el valor resultante de la expresión.

Versiones: Cassette, diskette.

Formato: ON <exprnum> {GOTO | GOSUB } <lista de nurolins>

Observaciones: El valor de <exprnum> determina el número de línea de la lista que será usado como destino del salto o desvío. Por ejemplo, si dicho valor es 3, el destino será el tercer número de línea de la lista. (Si el valor no es entero, se desprecia la parte fraccionaria).

Si el valor de <exprnum> es cero, o es mayor que el número de elementos mencionados en la lista (pero no menor ni igual a 255), BASIC continúa con la siguiente instrucción ejecutable. Si el valor de <exprnum> es negativo o mayor de 255, se produce un error de la clase 'cita ilegal de una función', y se detiene la ejecución.

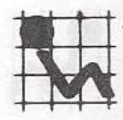
Ejemplo:

```
100 ON L GOTO 150, 200, 300
```

Si L es igual a 1, el programa salta a la línea 150; si L es igual a 2, a la línea 200; si L=3 a la línea 300.

```
100 ON M GOSUB 1000, 1500
```

Si M es igual a 1, se desvía a la subrutina situada a partir de la línea 1000; si M es igual a 2, a la situada a partir de la línea 1500.



4.1.1.30 POKE

Propósito: Escribir un carácter en la memoria.

Versiones: Cassette, diskette.

Formato: POKE <exprnumdir> , <exprnumcar>

Observaciones: <exprnum dir> es la dirección de la celdilla de memoria donde se va a meter; mientras que <exprnum car> es el carácter (octeto) que va a ser metido.

El valor que se mete ha de estar dentro de la gama 0 a 255; y la dirección de memoria debe estar en la gama -32768 a +65535.

(Teniendo en cuenta que cuando la dirección es un valor negativo, es equivalente a la dirección positiva que se obtiene al restarla de 65536).

Por ejemplo, -1 es la misma dirección que 65535 (=65536-1).

En caso de que no estén dentro de las gamas citadas, se produce un error de desbordamiento o rebase: **"Overflow"**.

Ejemplo: POKE & H5A00, &HFF
Escribe el dato 255 (&HFF) en la dirección 23040 (&H5A00).

4.1.1.31 PRINT

Propósito: Enviar datos hacia el monitor, para ser expuestos en pantalla.

Versiones: Cassette, diskette.

Formato: PRINT [<remesa de datos>]

Observaciones: Si se incluye <remesa de datos> , se exponen sucesivamente en la pantalla del monitor. Los datos de la remesa pueden ser constantes, variables, o expresiones numerales y literales. Las constantes literales deben encerrarse entre un par de comillas.

La posición de cada dato en la pantalla queda determinada por el signo de puntuación usado para separar los datos de la remesa. BASIC divide una línea de pantalla en **zonas** de exposición que ocupan 14 posiciones cada una. En la <remesa de datos> , cada coma hace que el siguiente dato a exponer aparezca al principio de la siguiente zona. Un punto y coma provoca que el siguiente dato a exponer aparezca inmediatamente detrás del último dato expuesto. Teclear uno o más espacios en blanco entre los datos de la remesa tiene el mismo efecto que teclear un punto y coma.

Si la remesa de datos a exponer termina con una coma o con un punto y coma, la siguiente instrucción PRINT comenzará a exponer sus datos en la misma línea, y separados acordemente; mientras que si termina sin una coma o sin un punto y coma, BASIC colocará un avance de línea y un retorno de carro al final de la línea expuesta.

Si los datos a exponer ocupan más de una línea a lo ancho de la pantalla, BASIC pasa a la siguiente renglón y continúa la exposición de datos. Cuando el dato es numérico, siempre va seguido de un espacio en blanco. Los números positivos van precedidos también por un espacio en blanco y los negativos por un signo menos.

Para abreviar se puede usar el signo de interrogación en lugar de la palabra PRINT en estas instrucciones.



Ejemplo:

```
10 PRINT "AREA DEL CIRCULO = 3.1416*
   RADIOS ^ 2"
20 PRINT
30 ? "RADIO"
40 INPUT R
50 A = 3.1416* R ^ 2
60 PRINT
70 PRINT "AREA IGUAL A"; A
RUN
AREA DEL CIRCULO = 3.1416* RADIO ^ 2
RADIO
?4

AREA IGUAL A 50.2656
Ok
```

La línea 10 manda que se exponga un literal. La línea 20 hace que se salte una línea. El signo de interrogación (?) puede usarse para sustituir la palabra PRINT. En la línea 70, se expone una constante literal y un número, sin que se avance a la zona siguiente entre ambos, a causa de haber usado el punto y coma para separarlos (;).



4.1.1.32 PRINT USING

Propósito: Exponer datos numerales y literales conformados según una determinada pauta.

Versiones: Cassette, diskette.

Formato: PRINT USING <conformatriz> ; <remesa de datos>

Observaciones: La <remesa de datos> consta de las expresiones literales o numerales que van a ser expuestas en pantalla, separadas por punto y coma.
La <conformatriz> es una variable o una constante literal constituida por caracteres especiales que conforman el aspecto que tendrán los datos en imagen; (actuando a manera de molde u **horma de edición en pantalla**). Estos caracteres conformadores (véanse más abajo) determinan la longitud, posiciones y símbolos que se mostrarán en el **campo de exposición** ocupado en pantalla por el correspondiente dato homólogo de la remesa a exponer.

Cuando los caracteres de la <conformatriz> corresponden a un dato **literal**, puede usarse uno de los tres caracteres siguientes:

!

Especifica un campo de una sola posición que será ocupado por la inicial del dato literal correspondiente.

Ejemplos:

```
A$="Japón"
Ok
PRINT USING "!";A$
J
Ok
```

(n blancos) \␣␣ .... ␣\

Especifica un campo de n + 2 posiciones que será ocupado por el dato literal correspondiente, empezando por la izquierda. Por lo tanto, si se usa en <conformatriz> simplemente \\ , sólo aparecerán los dos caracteres anteriores del dato; si se usa \ \ , aparecerán los tres primeros; y así sucesivamente.
Si el dato literal es mayor que el campo conformado, se descartan los últimos caracteres del dato.
Si el campo conformado es mayor que el dato literal, éste quedará alineado por la izquierda y relleno por la derecha con los espacios en blanco pertinentes.



Ejemplo:

```
A$="JAPONES"
Ok
PRINT USING "ESTO NO ES \     \ AMIGO";A$
ESTO NO ES JAPON AMIGO
Ok
```

& especifica un campo de longitud variable, i.e. la que tenga el dato literal correspondiente.

```
A$(0)="LARGO:A$(1)="CORTO: I = 0
Ok
PRINT "SOY TAN & COMO QUIERAS";A$(I)
SOY TAN LARGO COMO QUIERAS
Ok
```

Cuando en la instrucción PRINT USING se usa la <conformatriz> para dar forma a **numerales**, se pueden emplear los siguientes símbolos.

#

Se usa para especificar una posición en el campo que será ocupada por un dígito, y que siempre se rellenará. Si el número a exponer tiene menos dígitos que las posiciones especificadas, será ajustado por la derecha y precedido de espacios en blanco.
El punto decimal para separar la parte fraccionaria de un número, puede insertarse en cualquier posición del campo. Si en la conformatriz se especifica que un dígito preceda al punto decimal, aparecerá siempre un dígito en pantalla (aunque sea cero si es necesario). Los números se redondean siempre que lo exige la precisión del campo.

Ejemplo:

```
PRINT USING "###.##";10.2,2,3.456,.24,123.5
10.20 2.00 3.46 0.24123.50
Ok
```

+

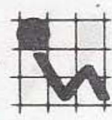
Un signo más al comienzo o al final de una horma hará que el signo del número (más o menos) se exponga antes o después del número, respectivamente.

Ejemplo:

```
PRINT USING "+###.##";1.25,-1.25
   +1.25 -1.25
Ok
PRINT USING "###.##+";1.25,-1.25
   1.25 + 1.25-
Ok
```

-

Un signo menos al final de la horma hará que los números negativos se muestren con un signo menos trasero.



Ejemplo:

```
PRINT USING "###.##-";1.25,-1.25
    1.25 1.25-
Ok
```

**

Un doble asterisco al comienzo de la horma provoca que los espacios delanteros de un campo numérico sean rellenos con asteriscos. Los asteriscos también especifican posiciones para dos o más dígitos del número a exponer.

Ejemplo:

```
PRINT USING "**#.##";1.25,-1.25
**1.25*-1.25
```

$$

Un doble dólar hace que aparezca el signo dólar inmediatamente a la izquierda del número expuesto en pantalla. El $$ especifica otras dos posiciones numerales más, una de las cuales es el signo dólar. No puede usarse la notación científica con $$. Los números negativos tampoco pueden usarse a no ser con el signo menos trasero.

Ejemplo:

```
PRINT USING "$$###.##";12.35,-12.35
     $12.35 -$12.35
Ok
PRINT USING "$$###.##-";12.35,-12.35
$12.35 $12.35-
Ok
```

**$

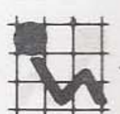
La inserción de **$ al comienzo de una horma de edición, combina los efectos mencionados para esos dos símbolos: los espacios en blanco serán rellenos con asteriscos y aparecerá un signo dólar antes del número. **$ especifica tres posiciones numerales más, una de las cuales es siempre el signo dólar.

Ejemplo:

```
PRINT USING "**$#.##";12.35
*$12.35
Ok
```

,

Una coma (,) situada a la izquierda de un punto decimal (.) en una horma de edición, hace que la coma aparezca como signo separador de la parte entera del número y cada tres cifras del mismo.



Una coma situada al final de la horma se expone como si fuera otro símbolo cualquiera de la propia horma. Una coma ocupa otra posición en el campo donde se expone el número. La coma no tiene ningún efecto si se usa la notación científica.

Ejemplo:

```
PRINT USING "####,.##";1234.5
1,234,50
Ok
PRINT USING "####.##,";1234.5
1234.50,
Ok
```

^^^^

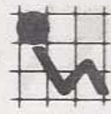
Se pueden situar cuatro **copetes** inmediatamente detrás de las posiciones numerales de la horma de edición, para especificar notación científica (exponencial), que serán ocupados por E±XX al mostrar el numeral correspondiente en pantalla. Se puede especificar cualquier posición para el punto decimal, y hará que se justifiquen por la izquierda los dígitos significativos del numeral y que se ajuste el exponente acordemente. A menos que se especifique en la horma un signo más delantero, o un signo más o menos trasero, una de las posiciones numerales situadas a la izquierda del punto decimal, quedará reservada para sacar un espacio en blanco cuando el número sea positivo o un signo menos cuando el número sea negativo.

Ejemplo:

```
PRINT USING "##.##^^^^";234.56
    2.35E+02
Ok
PRINT USING "#.##^^^^+";-12.34
    1.23E+01-
Ok
PRINT USING"+#.##^^^^";12.34,-12.34
    +1.23E+01-1.23E+01
Ok
```

%

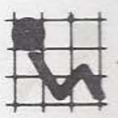
Si el número a exponer es mayor que el campo especificado por la horma correspondiente, se expondrá el signo de porcentaje delante del número expuesto. Además, si por los redondeos efectuados, el número sobrepasa la longitud del campo, también se expondrá el signo % delante del número redondeado.



Ejemplo:

```
PRINT USING "##.##";123.45
%123.45
Ok
PRINT USING ".##;.999
%1.00
Ok
```

Si el número de posiciones especificadas en una horma numeral es superior a 24, se producirá un error de 'cita ilegal de función'.



4.1.1.33 READ

Propósito: Apuntar en variables los valores constantes tomados correlativamente de las instrucciones DATA.

Versiones: Cassette, diskette.

Formato: READ <lista de variables>

Observaciones: Una instrucción READ guarda siempre relación con las instrucciones DATA que haya en el programa. Mediante las instrucciones READ se apuntan en las variables mencionadas en <lista de variables> , de una forma correlativa y secuencial las constantes mencionadas en las instrucciones DATA. Las variables de la instrucción READ pueden ser numéricas o litéricas, y los valores homólogos de las instrucciones DATA deben ser homogéneos con la variable correspondiente. Si no concuerdan, se producirá un 'error sintáctico'.

Una única instrucción READ puede apuntar valores tomados de una o más instrucciones DATA (y las instrucciones DATA siempre serán accesadas secuencialmente según su número de línea); o varias instrucciones READ pueden tomar los valores de la misma instrucción DATA. Si el número de variables mencionadas en <lista de variables> sobrepasa el número de elementos mencionados en las instrucciones DATA correspondientes, se producirá un error de 'escasez de datos' (Out of DATA). Si el número de variables mencionadas es menor que el número de elementos de las instrucciones DATA, las siguientes instrucciones READ comenzarán a tomar los valores a apuntar en sus variables a partir del primer elemento de las instrucciones DATA que no haya sido apuntado todavía. Y si no hay instrucciones READ subsiguientes, las constantes extra serán descartadas.

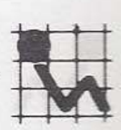
Para colocar el **puntero interno** que va 'punteando' los valores tomados en las instrucciones DATA, repunte hacia una instrucción DATA determinada, usa la instrucción RESTORE.

Ejemplo:

```
10 FOR I = 1 TO 3
20 READ NOME$(I), EDAD(I)
30 NEXT
40 DATA JULIO, 42, JULIA, 24, JULIAN, 21
```

Este programa apunta las constantes literales y numerales de la instrucción DATA de la línea 40, correlativamente como valores de las variables mencionadas en la línea 20. Si se tuviera que incluir un signo separador, tal y como dos-puntos, dentro de los nombres, la línea 40 se cambiaría para que fuera:

```
40 DATA "JULIO:", 42, "JULIA:", 24,
   "JULIAN:", 21
```

4.1.1.34 REM

Propósito: Incluir en un programa comentarios que sirvan de memorandum.

Versiones: Cassette, diskette.

Formato: REM <memorandum>

Observaciones: Las instrucciones REM no son ejecutables, sino que se reproducen exactamente cuando se lista el programa. las instrucciones REM pueden ser destino de saltos (mencionándolas en instrucciones GOTO o GOSUB), y la ejecución continuará con la primera instrucción ejecutable que venga detrás de la instrucción REM.

Los comentarios y memorandums pueden añadirse al final de una línea del programa, precediéndolos de un apóstrofe en lugar de la palabra REM; pero no pueden usarse en una instrucción DATA ya que serían considerados como constantes literales ilegales.

Ejemplo:

```
10 '
20 REM CALCULA DISTANCIA VIAJADA
30 INPUT "VELOCIDAD MEDIA" ; V
40 INPUT "TIEMPO EMPLEADO" ; T
50 D = V * T
60 PRINT "DISTANCIA RECORRIDA ES" ; D
RUN
VELOCIDAD MEDIA? 6
TIEMPO EMPLEADO? 8
DISTANCIA RECORRIDA ES 48
Ok
```

La línea 10 muestra que el apóstrofe (') produce el mismo efecto que la palabra REM. REM es muy útil para documentar programas, y sobre todo para colocarlas como aclaraciones al destino de los saltos y desvíos.



4.1.1.35 RESTORE

Propósito: Hacer que el 'puntero' que controla las instrucciones DATA señale a una cualquiera de ellas.

Versiones: Cassette, diskette.

Formato: RESTORE [<nurolin>]

Observaciones: Después de ejecutar una instrucción RESTORE, que no mencione un número de línea específico, la siguiente instrucción READ apunta en sus variables la primera constante que haya en la primera instrucción DATA del programa. Si se menciona <nurolin> lo toma de dicha instrucción DATA. Si el número de línea mencionado no existe en el programa, se producirá como resultado un error de 'número de línea sin definir'.

Ejemplo:

```
10  DIM X (12)
20  FOR I = 1 TO 3
30  FOR K = 1 TO 4
40  READ X (L)
50  PRINT X (L);
60  NEXT
70  PRINT
80  RESTORE
90  NEXT
100 DATA 1, 2, 3, 4
RUN
1   2   3   4
1   2   3   4
1   2   3   4
Ok
```

La instrucción RESTORE de la línea 80, hace que el puntero de instrucciones DATA repunte a la primera y única de ellas. Por lo tanto, vuelve a usar el programa los mismos DATOS que anteriormente.



4.1.1.36 RESUME

Propósito: Reanudar la ejecución del programa, después de haberse ejecutado la rutina señalada para tratamiento de errores.

Versiones: Cassette, diskette.

Formato: RESUME [0 | NEXT | <nurolin>]

Observaciones: El efecto depende de la opción utilizada:

RESUME o RESUME 0
La ejecución se reanuda en la propia instrucción que provocó el error.

RESUME NEXT
La ejecución se reanuda en la instrucción inmediatamente siguiente a la que provocó el error.

RESUME <nurolin>
La ejecución se reanuda en la línea mencionada por <nurolin> .

Una instrucción RESUME que se intente ejecutar sin haberse producido previamente un error, dará como resultado el mensaje de 'reanudación sin error previo'.

Ejemplo:

```
10  ON ERROR GOTO 100
20  FOR I = 1 TO 10
30  READ X(I)
40  NEXT
50  DATA 5, 4, 3, 2, 1
60  END
100 IF ERR=4 AND ERL=30 THEN PRINT
    "FALTAN DATOS"
110 RESUME 20
RUN
FALTAN DATOS
Ok
```

En la línea 100 comienza la rutina señalada para cuando ocurre algún error. La instrucción RESUME de la línea 110, dirige el programa para que continúe en la línea mencionada.



4.1.1.37 STOP

Propósito: Parar la ejecución del programa, haciendo que se regrese al nivel de comandos.

Versiones: Cassette, diskette.

Formato: STOP

Observaciones: La instrucción STOP se puede usar en cualquier parte de un programa para parar su ejecución. Cuando se encuentra la instrucción STOP, el sistema presenta en pantalla el siguiente mensaje:

Break in nnnn (nnnn es el número de línea)

A diferencia de la instrucción END, la instrucción STOP no cierra los ficheros.

La ejecución se puede continuar si se da el comando CONT.

Ejemplo:

```
10  FOR I = 1 TO 4
20  INPUT X
30  SUMA = SUMA + X
40  NEXT
50  STOP
60  PRINT "SUMA"; SUMA
70  END
RUN
? 4
? 5
? 6
? 7
Break in 50
Ok
CONT
SUMA = 22
Ok
```

Este ejemplo calcula la suma de 4 cifras, y luego se para en la línea 50; con el mensaje "Break in 50". Se usa CONT para que siga la ejecución del programa.



4.1.1.38 SWAP

Propósito: Canjear el valor de dos variables.

Versiones: Cassette, diskette.

Formato: SWAP <variable> , <variable>

Observaciones: Se pueden canjear entre sí los valores de dos variables cualesquiera (enteras, simple precisión, doble precisión, literales); pero obviamente, las dos variables deben tener la misma índole o se producirá un error de 'discordancia de clase'.

Ejemplo:

```
10  A$ = "CARA" : B$ = "Y" : C$ = "CRUZ"
20  PRINT A$; B$; C$
30  SWAP A$, C$
40  PRINT A$ + B$ + C$
RUN
CARA Y CRUZ
CRUZ Y CARA
Ok
```

La línea 30 hace que se intercambie el contenido de la variable A$ con el de la variable C$.

4.1.2 Funciones intrínsecas, excepto entrada/salida

Llamamos funciones a aquellos recursos intrínsecos del lenguaje de una máquina que al ser citados entregan como resultado de su acción un valor dependiente de los argumentos designados al mencionar la función en un programa.

4.1.2.1 ABS

Propósito: Extraer el valor absoluto de un argumento.

Versiones: Cassette, diskette.

Formato: ABS (<exprnum>)

Observaciones: <exprnum> es el argumento de la función y puede ser cualquier expresión numérica. El resultado dado por la función es el valor absoluto del argumento, y siempre será positivo o cero.

Ejemplo:

```
PRINT ABS (-5 * . 325)
 1.625
Ok
```

4.1.2.2 ASC

Propósito: Obtener el código ASCII que corresponde al primer carácter de una expresión literal.

Versiones: Cassette, diskette.

Formato: ASC (<exprlit>)

Observaciones: El resultado de la función ASC es un valor numérico entero que refleja el código ASCII del primer carácter del argumento <exprlit>. Si el argumento es nulo se produce un error de 'cita ilegal de función'.

La función CHR$ es la inversa de la función ASC y da como resultado el carácter correspondiente al código ASCII.

Consulta el Apéndice "Códigos y caracteres ASCII".

Ejemplo:

```
10 X$ = "TEST"
20 PRINT ASC (X$)
RUN
84
Ok
```

Este ejemplo muestra que el código ASCII para el carácter simbolizado por la letra "T" es 84.

4.1.2.3 ATN

Propósito: Obtener el arco cuya tangente viene dada por el argumento.

Versiones: Cassette, diskette.

Formato: ATN (<exprnum>)

Observaciones: El argumento <exprnum> puede ser cualquier numeral de cualquier clase; pero la función se evalúa siempre en doble precisión. El resultado viene expresado en radianes y dentro de la gama $-\pi/2$ a $+\pi/2$.

Para convertir radianes a grados, basta multiplicar por el valor $180/\pi$ siendo $\pi = 3.141593$

Ejemplo:

```
10  PI = 3.141593#
20  RADIANES ATN (6/8)
30  GRADOS = RADIANES * 180/PI
40  PRINT "TG#";
50  PRINT USING "###.##"; GRADOS;
60  PRINT ") = 6/8"
RUN
TG# (36.87) = 6/8
Ok
```

4.1.2.4 BIN$

Propósito: Obtener la representación binaria de un número en base 10.

Versiones: Cassette, diskette.

Formato: BIN$ (<exprnum entera>)

Observaciones: <exprnum entera> ha de estar en la gama -32768 a +65535. Si no es entera, se desprecia su parte fraccionaria. Si es negativa, se usa la forma de complemento a doses; por lo tanto, BIN$(-n) es lo mismo que BIN$ (65536 - n).

Ejemplo:

```
PRINT BIN$ (12)
1100
Ok
```

Este ejemplo muestra que el número 12 en base 10 es igual que el número binario 1100.



4.1.2.5 CDBL

Propósito: Convierte una expresión numérica en un numeral de doble precisión.

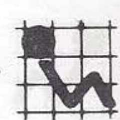
Versiones: Cassette, diskette.

Formato: CDBL (<exprnum>)

Observaciones: <exprnum> es cualquier expresión numérica. Consulta las funciones CINT y CSNG para convertir expresiones numéricas en numerales enteros y de simple precisión respectivamente.

Ejemplo:

```
PRINT CDBL (34/7)
4.8571428571429
Ok
```

4.1.2.6 CHR$

Propósito: Obtener el carácter equivalente a un código ASCII.

Versiones: Cassette, diskette.

Formato: CHR$ (<exprnum entera>)

Observaciones: <exprnum entera> ha de estar en la gama de 0 a 255. El resultado de esta función es un literal con un solo carácter que corresponde al código ASCII del argumento de la función. Se utiliza habitualmente para enviar caracteres especiales hacia la pantalla o la impresora.

La función inversa que obtiene el código ASCII correspondiente a un determinado carácter, es la función ASC.

Ejemplo:

```
PRINT CHR$ (85)
U
Ok
```

Porque el código ASCII que corresponde al carácter simbolizado con la letra "U" es 85.

4.1.2.7 CINT

Propósito: Convertir el argumento a un numeral entero.

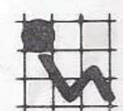
Versiones: Cassette, diskette.

Formato: CINT (<exprnum>)

Observaciones: <exprnum> puede ser cualquier expresión numérica, y el resultado de la función es el valor de esa expresión considerado como entero, obtenido redondeando la parte fraccionaria si la hubiera, y dentro de la gama -32768 a +32767.

Ejemplo:

```
PRINT CINT (45.67)
46
Ok
```

4.1.2.8 COS

Propósito: Obtener el coseno de un ángulo expresado en radianes.

Versiones: Cassette, diskette.

Formato: COS (<exprnum>)

Observaciones: El argumento <exprnum> es el ángulo cuyo coseno calcula la función, y debe estar expresado en radianes. Para pasar de grados a radianes, multiplica éstos por π/180, siendo π=3.141593.

El cálculo de las funciones trigonométricas se efectúa en doble precisión.

Ejemplo:

```
PRINT COS (2)
-.4161468365472
Ok
```



4.1.2.9 CSNG

Propósito: Convertir a numeral de simple precisión cualquier expresión numérica.

Versiones: Cassette, diskette.

Formato: CSNG (<exprnum>)

Observaciones: <exprnum> es cualquier expresión numérica. Consulta las funciones CINT y CDBL para convertir a enteros y doble precisión, respectivamente.

```
10  A = 345.53454663
20  PRINT CSNG (A)
RUN
345.535
Ok
```

4.1.2.10 CSRLIN

Propósito:	Da el valor asociado a la línea horizontal donde está ubicado el cursor.
Versiones:	Cassette, diskette.
Formato:	CSRLIN
Observaciones:	La variable CSRLIN tiene como valor la línea horizontal vigente ocupada por el cursor en la página activa. El valor de esa variable está en la gama 1 a 25.

Consulta la función POS para hallar la columna donde está ubicado el cursor, dentro de la línea designada por CSRLIN.

Consulta la instrucción LOCATE para ver cómo se ubica el cursor.

Ejemplo:

```
10 CLS
20 LOCATE 20, 5
30 A = POS (0)
40 B = CSRLIN
50 PRINT A; B
```

En este ejemplo, el cursor se ubica dentro de la línea horizontal 20, en la columna número 5. Luego, las coordenadas del cursor se registran como valores de las variables A y B.



4.1.2.11 ERL/ERR

Propósito: Reflejar respectivamente el número de línea y el código de error correspondientes al error producido durante la ejecución.

Versiones: Cassette, diskette.

Formato: ERL
ERR

Observaciones: Cuando se produce un error, la variable del sistema ERR contiene el código asociado a ese error, y la variable ERL el número de línea en que se detectó el error. Habitualmente estas variables se utilizan en instrucciones IF...THEN para dirigir la secuencia del programa, dentro de la rutina de tratamiento de errores. Si se comprueba ERL en una instrucción IF...THEN, debes hacerlo en la forma:

IF ERL = número-de-línea THEN...

para que pueda ser modificado acordemente cuando ejecutes el comando RENUM.

Si la instrucción que provoca el error fue un comando (dado en modo directo), la variable ERL tendrá como valor 65535. Para comprobar si se produjo un error en un comando de modo directo, úsala mejor en la forma 65535=ERL THEN...; porque en caso contrario, te daría error al hacer la operación de renumeración.

Ejemplo:

```
10  ON ERROR GOTO 100
20  INPUT "NUMERO"; N
30  IF N >100 THEN ERROR  61
40  END
100 IF (ERR = 61) AND (ERL = 30) THEN
    PRINT "DEMASIADO": RESUME 20
NUMERO ? 789
DEMASIADO
NUMERO ? 7
Ok
```

En este ejemplo, se establece una rutina para el tratamiento de errores que pueda producirse al teclear un número. La instrucción RESUME de la línea 100 hace que el programa vuelva a efectuar la línea 20 cuando se detecta el error 61 en la línea 30.



4.1.2.12 EXP

Propósito: Calcular la función exponencial.

Versiones: Cassette, diskette.

Formato: EXP (<exprnum>)

Observaciones: El argumento <exprnum> es cualquier expresión numérica, y ha de ser menor o igual que 145.06286085862.

Esta función da como resultado el número e elevado a la potencia <exprnum> . El número e es la base de los logaritmos naturales, y esta función también se denomina antilogaritmo. Si el argumento rebasa la cota establecida, se produce un error de 'desbordamiento' o rebase.

Ejemplo:

```
PRINT EXP (2)
7.38905609893
Ok
```



4.1.2.13 FIX

Propósito: Obtener la parte entera ('fija') de un numeral.

Versiones: Cassette, diskette.

Formato: FIX (<exprnum>)

Observaciones: El argumento es cualquier expresión numérica, y su parte entera -simplemente quitando la parte fraccionaria- es el resultado de la función. Se verifica que

FIX (X) = SGN (X) *INT (ABS (X))

La mayor diferencia entre las funciones FIX y SGN, estriba en que FIX no da como resultado el siguiente número inferior para argumentos negativos.

Ejemplo:

```
10  PRINT "INT (-34.5) =”; INT (-34.5)
20  PRINT "FIX (-34.5) =”; FIX (-34.5)
30  PRINT "SGN (-34.5) *INT (ABS (-34.5))
RUN
INT (-34.5) = -35
FIX (-34.5) = -34
SGN (-34.5)* INT (ABS (-34.5)) = -34
Ok
```

4.1.2.14 FRE

Propósito: Hallar el número de **octetos** de memoria que no están siendo usados por BASIC.

Versiones: Cassette, diskette.

Formato: FRE ({<exprnum> | <exprlit>})

Observaciones: Los argumentos de esta función son 'postizos' (también llamados ficticios, porque realmente no actúan como tal).

Cuando el argumento es numérico, el valor obtenido corresponde a los octetos de memoria que pueden usarse para programas BASIC, ficheros de texto, programas en lenguaje máquina, etc. Cuando el argumento es litérico, el resultado corresponde al número de octetos en memoria ocupados por constantes literales.

Ejemplo:

```
10 GOSUB 80
20 DIM A (100) : DIM A$ (100)
30 FOR I = 1 TO 100
40 A(I) = I : A$ (I) = CHR$ (I)
50 NEXT
60 GOSUB 80
70 END
80 PRINT "FRE (0) =" FRE (0),
90 PRINT "FRE (" + CHR$ (34) + CHR$
   (34) +") = "FRE ("")
100 PRINT
110 RETURN
RUN
FRE (0) = 29025 FRE ("") = 200 FRE (0)
= 27887 FRE ("") = 100
Ok
```



4.1.2.15 HEX$

Propósito: Obtener el valor literal que corresponde en el sistema hexadecimal a un argumento en base 10.

Versiones: Cassette, diskette.

Formato: HEX$ (<exprnum entera>)

Observaciones: El argumento <exprnum entera> ha de estar en la gama -32768 a +65535; y si no es un valor entero, se prescinde de su parte fraccionaria. Si es negativo, se usa la forma de complemento a doses. Por lo tanto, HEX$(-n) es lo mismo que HEX$(65536-n).

Ejemplo:

```
PRINT HEX$(-32768), HEX$(65535)
8000          FFFF
Ok
```

Este ejemplo usa la función HEX$ para obtener la representación en hexadecimal de los dos valores en base 10 que se han mencionado.

4.1.2.16 INKEY$

Propósito: Tomar un solo carácter del teclado.

Versiones: Cassette, diskette.

Formato: INKEY$

Observaciones: El resultado puede ser un literal de un solo carácter correspondiente al que estuviera pulsado en el teclado, o la 'cadena vacía' cuando no hay ninguna tecla pulsada. Los caracteres no serán 'repicados' en pantalla, y cualquiera que sea el pulsado será traspasado directamente al programa, exceptuando CONTROL y C que termina el programa en el acto.

Ejemplo:

```
10 PRINT "PULSA CUALQUIER TECLA
   PARA CONTINUAR"
20 A$ = INKEY$
30 IF A$ = "" THEN 20
40 CLS
```

Esta sección del programa suspende la ejecución, manteniéndose en un bucle, hasta que se pulsa cualquier tecla en el teclado.

4.1.2.17 INPUT$

Propósito: Ingresar un dato literal de una longitud predeterminada.

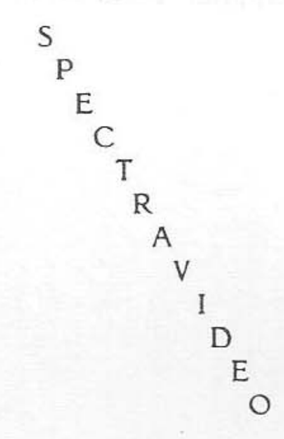
Versiones: Cassette, diskette.

Formato: INPUT$ (<exprnum1> [, [#] <exprnum2>])

Observaciones: El argumento <exprnum1> determina el número de caracteres que serán aceptados del teclado. Ninguno de ellos será 'repicado' en pantalla, y todos serán traspasados al programa, exceptuando CONTROL y C que termina la ejecución de la función INPUT$.

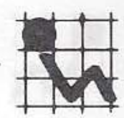
Ejemplo:

```
10 PRINT "TECLEA UN LITERAL DE
   DOCE CARACTERES"
20 X$ = INPUT $ (12)
30 IF X $ = "" THEN 20
40 PRINT
50 FOR I = 1 TO 12
60 PRINT TAB(I+10); MID$ (X$, I 1)
70 NEXT
RUN
TECLEA UN LITERAL DE DOCE CARACTERES
```




```
Ok
```

La línea 20 espera que se ingresen mediante el teclado los 12 caracteres mencionados.



4.1.2.18 INSTR

Propósito: Escrutar un determinado literal para ver si en él aparece otro literal dado. La búsqueda puede opcionalmente estipularse que comience a partir de una determinada posición.

Versiones: Cassette, diskette.

Formato: INSTR ([<exprnum1> ,] <exprlit1> , <exprlit2>)

Observaciones: <exprnum1> ha de ser un numeral entero en la gama 0 a 255. Los argumentos <exprlit1> y <exprlit2> pueden ser variables, constantes o expresiones literales. Si el número de caracteres del literal buscado es menor que la longitud del literal donde se busca, o si éste es nulo, o si no se encuentra el literal buscado dentro del otro literal, o si ambos son nulos, el valor entregado por la función INSTR es cero.

Si sólo <exprlit2> es nulo, INSTR entrega como valor el de <exprnum1> , o bien el 1 cuando no se menciona el anterior.

Ejemplo:

```
10  A$ = "GUANTANAMERA"
20  B$ = "AN"
30  PRINT INSTR (A$, B$); INSTR (5, A$, B$)
RUN
3       6
Ok
```

Este ejemplo escruta el literal A$, para ver si en él aparece el literal B$. Cuando en la primera parte de la línea 30, se le pide que examine A$, a partir de la primera posición, lo encuentra y se detiene dándonos el valor 3; en el segundo caso de la línea 30, se le pide que comience su exámen a partir de la posición 5, por lo que nos devuelve el valor 6 para indicar que es la posición ocupada por B$ dentro del literal A$.



4.1.2.19 INT

Propósito: Obtener el mayor de los numerales enteros que es menor o igual que un numeral dado.

Versiones: Cassette, diskette.

Formato: INT (<exprnum>)

Observaciones: El argumento <exprnum> es cualquier expresión numérica. Compara con las funciones FIX y CINT que también dan como resultado números enteros.

Ejemplo:

```
PRINT INT (45.6)
45
Ok
```

Dado que 45.6 = 45 + 0.6; se deduce que 45 es el mayor entero que es menor que 45.6

```
PRINT INT(-45.6)
-46
Ok
```

Dado que -45.6 = -46 + 0.4; se deduce que -46 es el mayor entero inferior a -45.6.

Nota: Se dice que esta función obtiene un número entero con redondeo por defecto.

4.1.2.20 LEFT$

Propósito: Segregar los caracteres anteriores -situados a la izquierda- de un literal.

Versiones: Cassette, diskette.

Formato: LEFT$ (<exprlit>, <exprnum>)

Observaciones: El valor devuelto es una constante literal formada por tantos caracteres de <exprlit> , como indique <exprnum> , y tomando los primeros o situados a la izquierda.

El argumento <exprnum> ha de ser entero y en la gama 0 a 255. Si su valor es mayor que el número total de caracteres que compone <exprlit> el valor literal obtenido es exactamente <exprlit> . Si el valor de <exprnum> es cero, el valor devuelto por la función es la 'cadena vacía' (cuya longitud es cero).

Ejemplo:

```
PRINT LEFT$ ("ANTERIORESMEDIANEROS
ULTERIORES", 10)
ANTERIORES
Ok
```

En este ejemplo, la función LEFT$ segrega del argumento literal los primeros 10 caracteres de la izquierda.



4.1.2.21 LEN

Propósito: Saber el número de caracteres que componen un literal dado.

Versiones: Cassette, diskette.

Formato: LEN (<exprlit>)

Observaciones: Hay 22 caracteres en el argumento literal, y recuerda que el 'blanco' es también un carácter.

Ejemplo:

```
PRINT LEN ("FRASES SESQUIPEDALICAS")
22
Ok
```

Resultan 22 caracteres en la cadena "FRASES SESQUIPEDALICAS" porque también se cuentan los espacios en blanco.



4.1.2.22 LOG

Propósito: Calcular el logaritmo natural de un número.

Versiones: Cassette, diskette.

Formato: LOG (<exprnum>)

Observaciones: El argumento <exprnum> debe ser un número positivo y mayor de 0. Los logaritmos naturales tienen como base el número e; y puedes consultar la función EXP que es la inversa de ésta.

Ejemplo:

```
PRINT LOG(3)
3.1345942159291
Ok
```



4.1.2.23 LPOS

Propósito: Saber la posición corriente del cabezal de la impresora, pero tal como **aparenta** dentro de la zona de memoria reservada para el envío de datos a la impresora.

Versiones: Cassette, diskette.

Formato: LPOS (<exprnum>)

Observaciones: El argumento de esta función es totalmente ficticio, pero ha de tener un valor entero.

Recuerda que la función LPOS no da necesariamente la posición que en la práctica tenga el cabezal de impresión.

Ejemplo:

```
IF LPOS(0) > 40 THEN LPRINT CHR$ (13)
```

En este ejemplo, si la posición 'aparente' del cabezal de impresión está situada detrás de la columna 40, se envía hacia la impresora el carácter cuyo código es 13 (que corresponde en el código ASCII al 'retorno de carro').

4.1.2.24 MID$

Propósito: Segregar una determinada porción de un literal.

Versiones: Cassette, diskette.

Formato: MID$ (<exprlit>, <exprnum1> [,<exprnum2>])

Observaciones: El argumento <exprlit> es cualquier expresión literal, de la que se va a segregar la porción.

El argumento <exprnum1> es cualquier expresión entera en la gama 1 a 255; e indica la posición a partir de la cual empiezan a extraerse caracteres. Si su valor es mayor que el número total de caracteres en el argumento literal, la función MID$ entrega como valor la 'cadena vacía'.

El argumento optativo <exprnum2> indica el número de caracteres a segregar. Si se omite, o si es menor que los caracteres que quedan a la derecha de la posición indicada por <exprnum1>, se obtiene como valor de MID$ un sub-literal formado por los caracteres que quedan hasta el final de <exprlit>.

Ejemplo:

```
X$ = "ANTERIORESMEDIANEROSULTERIORES"
Ok
PRINT MID$ (X$, 11, 10)
MEDIANEROS
Ok
```

El segundo de los comandos expone una cadena de 10 caracteres, tomados a partir del 11º carácter del literal X$.



4.1.2.25 OCT$

Propósito: Obtener la representación en el sistema de base 8 de un numeral dado en el sistema de base 10.

Versiones: Cassette, diskette.

Formato: OCT$ (<exprnum entera>)

Observaciones: El argumento es cualquier expresión numeral entera, en la gama -32768 a +65535. Si es negativo, se utiliza la notación de complemento a doses; es decir, OCT$(-n) es lo mismo que OCT$(65536-n).

Ejemplo:

```
PRINT OCT$ (-32768), OCT$ (65535)
100000              177777
Ok
```

4.1.2.26 PEEK

Propósito: Entregar el valor decimal equivalente al contenido de una determinada celdilla de memoria.

Versiones: Cassette, diskette.

Formato: PEEK (<exprnum entera>)

Observaciones: El argumento puede ser cualquier expresión numérica entera, dentro de la gama -32768 a +65535, y representa la dirección de la celdilla cuyo contenido se quiere examinar.

PEEK es una función que complementa al comando POKE.

Ejemplo:

```
10 POKE &H9C40, 5
20 PRINT PEEK(&H9C40); PEEK (%0116100);
   PEEK (&B1001110001000000)
RUN
5  5  5
Ok
```

El ejemplo muestra diferentes formas de dar la dirección en memoria cuyo contenido se quiere examinar. En el ejemplo, se da según el sistema hexadecimal, octal y binario.



4.1.2.27 POS

Propósito: Saber la posición corriente del cursor según el eje horizontal.

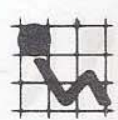
Versiones: Cassette, diskette.

Formato: POS (<exprnum>)

Observaciones: El argumento de la función es ficticio, aunque debe tener un valor numeral legal. La posición más a la izquierda en la pantalla corresponde al número 0. Consulta la función CSRLIN para lo referente a la ubicación del cursor según el eje vertical.

Ejemplo:

```
IF POS(0) > 60 THEN PRINT CHR$ (13)
```

4.1.2.28 RIGHT$

Propósito: Segregar los caracteres ulteriores -los colocados a la derecha- de un literal determinado.

Versiones: Cassette, diskette.

Formato: RIGHT$ (<exprlit>, <exprnum>)

Observaciones: El argumento <exprnum> ha de ser entero y en la gama 0 a 255; y especifica el número de caracteres a segregar para formar el literal resultante. Si su valor es igual al número de caracteres que hay en el argumento <exprlit> , el resultado coincide con el argumento literal. Si su valor es 0, el resultado es la 'cadena vacía'.

Ejemplo:

```
10  X$ = "ANTERIORESMEDIANEROS
    ULTERIORES"
20  PRINT RIGHT$ (X$, 10)
RUN
ULTERIORES
Ok
```

En el ejemplo se sacan los 10 caracteres últimos -los situados a la derecha- del literal X$.

4.1.2.29 RND

Propósito: Obtener un número aleatorio entre 0 y 1.

Versiones: Cassette, diskette.

Formato: RND (<exprnum>)

Observaciones: El argumento <exprnum> es una expresión numérica cuyo valor determina el resultado de la función.

Cada vez que se ejecuta un programa, se genera la misma secuencia de números aleatorios, a no ser que se use un valor diferente para el argumento de esta función cada vez. Si el valor de <exprnum> es menor que 0, el generador de números aleatorios, utiliza siempre el mismo 'germen' para generar la secuencia.

Si el valor del argumento es 0, se repite siempre el último número generado. Si el valor del argumento es menor que 0 el resultado es el siguiente número aleatorio que exista en la secuencia.

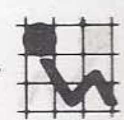
Ejemplo:

```
10  FOR I = 1 TO 2
20  PRINT RND (2);
30  NEXT
40  PRINT: PRINT RND (0)
50  FOR I = 1 TO 2
60  PRINT RND(-2);
70  NEXT
RUN
     .59521943994623          .10658628050158
     .10658628050158
     .94389820420821          .94389820420821
Ok
```

La primera fila muestra dos números aleatorios sucesivos, generados al usar un argumento positivo en la función RND.

En la línea 40, la función RND se cita con un argumento igual a cero, por lo que el número generado es exactamente el mismo que se generó más recientemente.

En la línea 60, el argumento es un número negativo, por lo que se vuelve a establecer el 'germen' a partir del que se genera la secuencia de números aleatorios.



4.1.2.30 SGN

Propósito: Saber el signo de una expresión numérica.

Versiones: Cassette, diskette.

Formato: SGN (<exprnum>)

Observaciones: El argumento es cualquier expresión numérica, y el resultado de la función es:

=1 cuando el argumento es positivo (>0)
=0 si el argumento es igual cero (=0)
=-1 si el argumento es menor que cero (< 0)

Ejemplo:

```
ON SGN(X) + 2 GOTO 100, 200, 300
```

Esta instrucción dirige el curso del programa hacia la instrucción 100 si X es negativo, hacia la 200 si es cero, y hacia la 300 si es positivo.

4.1.2.31 SIN

Propósito: Calcular el seno de un ángulo dado en radianes.

Versiones: Cassette, diskette.

Formato: SIN (<exprnum>)

Observaciones: El argumento de la función es cualquier expresión numérica cuyo valor viene interpretado como los radianes del ángulo cuyo seno se desea calcular. Para convertir grados a radianes, multiplica por π/180, siendo π=3.141593. El resultado se calcula en doble precisión.

Ejemplo:

```
PRINT SIN(0)
0
Ok
```

En el ejemplo, el ángulo cuyo seno se calcula tiene 0 radianes.

4.1.2.32 SPACE$

Propósito: Formar un literal con un determinado número de espacios en blanco.

Versiones: Cassette, diskette.

Formato: SPACE$ (<exprnum entera>)

Observaciones: Si el argumento no es un numeral entero, se quita la parte fraccionaria y se forman tantos blancos como indique el valor resultante, que ha de estar en la gama 0 a 255.

Consulta también la cláusula SPC, usada en instrucciones PRINT.

Ejemplo:

```
10  PRINT " "
20  FOR I = 1 TO 5
30  X$ = SPACE$ (I)
40  PRINT X$: "L"
50  NEXT
RUN

_
 L
  L
   L
    L
     L
Ok
```

En este ejemplo se usa la función SPACE$ para exponer el carácter "L" en una línea precedido por I espacios en blanco.



4.1.2.33 SPC

Propósito: Usado siempre como cláusula de un comando PRINT o LPRINT, para exponer un determinado número de espacios en blanco.

Versiones: Cassette, diskette.

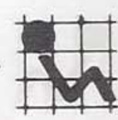
Formato: SPC (<exprnum entera>)

Observaciones: El argumento ha de estar en la gama 0 a 255, y estipula el número de blancos que han de exponerse por monitor o por impresora. La cláusula SPC tiene **implícito** un punto y coma después de ella.

Ejemplo:

```
PRINT "SEPARO" SPC(5) "CON BLANCOS"
SEPARO ↑↑↑↑↑ CON BLANCOS
```

Observa que hay cinco espacios en blanco entre los dos literales expuestos en pantalla.



4.1.2.34 SQR

Propósito: Extraer la raíz cuadrada de un número.

Versiones: Cassette, diskette.

Formato: SQR (<exprnum positiva>)

Observaciones: El argumento es cualquier expresión numérica, siempre que su valor sea **positivo.**

```
PRINT SQR (25)
5
Ok
```

La raíz cuadrada de 25 es 5.

4.1.2.35 STR$

Propósito: Pasar a representación literal un dato numeral.

Versiones: Cassette, diskette.

Formato: STR$ (<exprnum>)

Observaciones: En realidad, el argumento aunque puede ser cualquier expresión numérica, es evaluado antes de ser transformado en literal. Observa que cuando está considerado como un literal, siempre se reserva un espacio en blanco delantero para el posible signo.

La función VAL es complementaria de STR$.

Ejemplo:

```
PRINT STR$ (8*7); LEN (STR$ (8*7))
 56     3
Ok
```

Ocho por siete da cincuenta y seis, y es evaluado antes de ser transformado en literal. Observa cómo la longitud del literal es 3.

4.1.2.36 STRING$

Propósito: Formar un literal de una determinada longitud cuyos caracteres son todos iguales a uno dado, bien por su código ASCII, bien por el primer carácter de otro literal.

Versiones: Cassette, diskette.

Formato: STRING$ (<exprnum1>, {<exprnum2> | <exprlit>})

Observaciones: El primer argumento numérico ha de ser entero y en la gama 0 a 255, e indica el número de caracteres que van a formar el resultado de la función.

El segundo argumento puede ser un entero que indica el código ASCII del carácter a utilizar, o una expresión literal cuyo primer carácter es el que formará el literal resultado de la función.

Ejemplo:

```
10  X$ = "PARA LO QUE VALGO"
20  Y$ = STRING$ (5, 42)
30  PRINT Y$ + STRING$ (5, X$) + Y$
RUN
*****PPPPP*****
Ok
```

En la línea 20, se forma un literal que consta de cinco asteriscos (código 42). En la línea 30, se forma y expone en pantalla un literal con cinco caracteres iguales al primero de la variable literal X$.



4.1.2.37 TAB

Propósito: Es siempre una cláusula de un comando PRINT o LPRINT, usada para hacer que se coloque el cursor o el cabezal de la impresora en una determinada columna.

Versiones: Cassette, diskette.

Formato: TAB (<exprnum entera>)

Observaciones: El argumento debe ser una expresión numérica entera en la gama 0 a 255, y el BASIC enviará al monitor o a la pantalla tantos blancos como sea necesario para colocar el cursor en la columna estipulada por el valor de <exprnum entera> .

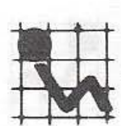
Si ya la posición corriente del cursor en el monitor o en la impresora ha sobrepasado la columna señalada por el valor de <exprnum entera> ; la cláusula TAB coloca el cursor en la siguiente línea.

La columna cero corresponde a la posición más a la izquierda, mientras que la posición a la derecha tiene como número de columna la anchura del monitor o de la impresora menos uno.

Ejemplo:

```
10  A$ = "OJO": B= "TABULACION"
20  PRINT A$ TAB (10) B$
30  PRINT A$ SPC (7) B$
RUN
OJO          TABULACION
OJO          TABULACION
Ok
```

La línea 20 manda exponer el valor de B$ a partir de la columna 10. Observa que el mismo efecto se produce usando la función SPC, tal y como hemos hecho en la línea 30. (Pero es que ya sabemos que "OJO" tiene 3 caracteres de longitud).



4.1.2.38 TAN

Propósito: Hallar la tangente de un ángulo dado en radianes.

Versiones: Cassette, diskette.

Formato: TAN (<exprnum>)

Observaciones: El argumento de la función mide el ángulo cuya tangente se quiere calcular en radianes. Para convertir grados a radianes, multiplica por π/180, siendo π=3.141593. El resultado se calcula en doble precisión.

Ejemplo:

```
PRINT TAN(0)
0
Ok
```

En este ejemplo se calcula la tangente de un ángulo de 0 radianes.



4.1.2.39 USR

Propósito: Ceder el control de la máquina a una subrutina en lenguaje máquina determinada, a la que se entrega un argumento numérico cualquiera y de la que se obtiene un resultado dependiente de la acción llevada a cabo por la subrutina.

Versiones: Cassette, diskette.

Formato: USR[<dígito>] (<exprnum>)

Observaciones: El parámetro opcional <dígito> permite elegir una entra las 10 posibles rutinas de usuario, y ha de corresponder al dado en la instrucción DEF USR que especifica la dirección donde comienza dicha subrutina. Si se omite, se adopta la correspondiente a USR0.

El argumento <exprnum> es una expresión que ha de ser acorde con lo que el usuario haya estipulado en la subrutina de lenguaje máquina.

Ejemplo:

```
10 DEF USR0 = &HFF30 : 'CALCULA
   RAICES CUBICAS 15 B=216
20 C = USR0 (B/8)
30 D = USR0 (B/29)
40 PRINT C, D
```

La función USR0 está DEFinida en la línea 10, y podemos suponer que calcula raíces cúbicas. En la línea 20 se cita la función USR0 con el argumento B/2, con lo que se cede el control a dicha subrutina en lenguaje máquina, que en este caso devolvería un valor (con el que dotamos a la variable C). En la línea 30 se le transfiere otro argumento, y el resultado es el valor de la variable D. (Si en esa dirección hubiera una subrutina que calculara raíces cúbicas, el valor de C sería 3 y el de D 2.



4.1.2.40 VAL

Propósito: Convertir una expresión literal en un valor numérico.

Versiones: Cassette, diskette.

Formato: VAL (<exprlit>)

Observaciones: El argumento de la función es cualquier expresión literal. El resultado de la función VAL es un número formado por los primeros caracteres numéricos que existan en <exprlit> . No se consideran los blancos delanteros, las marcas de tabulación o de avances de línea. Se detiene la conversión cuando se encuentra el primer carácter que no sea numérico; por lo tanto, si el primero ya no es numérico, el resultado de la función VAL será el número 0.

Consulta la función STR$ para hacer la conversión inversa.

Ejemplo:

```
PRINT VAL ("4.20 TARIFA")
 4.2
Ok
```

La función VAL sólo toma la primera parte que puede considerarse como numérica en el argumento literal. Se incluye un espacio delantero para el posible signo, y se quitan todos los caracteres que estén situados a partir del primero que no sea numérico.

Si el argumento literal empieza por lo que puede considerarse como una expresión numérica, la función VAL no la calcula.



4.1.2.41 VARPTR

Propósito: Saber la dirección en memoria donde está alojada una determinada variable, o un bloque de control de ficheros.

Versiones: Cassette, diskette.

Formato: VARPTR ({<nomevar> | #<nurofich>)

Observaciones: Cuando se usa como argumento <nomevar> es el nombre de una variable numérica o litérica, o de un elemento de una tabla, cuya dirección en memoria es el resultado de la función VARPTR. Al citar la función, debe asignarse un valor a <nomevar> antes de usar la función, porque de lo contrario daría como resultado un error de 'cita ilegal de función'.

Cuando se usa como argumento # <nurofich> debe corresponder al número con el que se ha abierto un fichero, y el resultado es la dirección donde comienza el área de memoria reservada para ese fichero.

La función VARPTR se utiliza habitualmente para obtener el **puntero** de una variable usada en el programa, para que pueda ser transferida a una subrutina en lenguaje máquina. La dirección entregada por la función será un entero en la gama -32768 a +32767; y si es negativo, basta añadirle 65536 para obtener la dirección en la forma positiva habitual.

Cuando se pasa la dirección de una tabla, se hace habitualmente una cita a la función en la forma VARPTR (A (0)), de manera que se obtenga la dirección del elemento más bajo de esa tabla. Todas las variables simples deberán estar asignadas antes de citar la función VARPTR para una tabla, porque la dirección de la tabla cambia siempre que se da valor a una nueva variable simple de las que componen la tabla.

Ejemplo:

```
10 A$ = "SUPERLATIVO"
20 B = VARPTR(A$)
30 PRINT HEX$ (B)
RUN
8033
```




4.2 INSTRUCCIONES Y FUNCIONES ESPECIFICAS DE LOS PERIFERICOS

4.2.1 Instrucciones

Comentaremos en esta sección la serie de instrucciones disponibles en el BASIC SV para intercambiar información con los equipos periféricos, tales como teclado, pantalla y archivos de almacenamiento.

4.2.1.1 SCREEN

Propósito: Fijar el modo de operación de pantalla y, opcionalmente, el recuadro ocupado por las figuras definidas mediante SPRITE$.

Versiones: Cassette, diskette.

Formato: SCREEN [<modo>] [,<recuadro>]

Observaciones: El <modo> puede ser:

0:	40 x 24	modo texto
1:	256 x 192	modo de alta resolución
2:	64 x 48	modo de baja resolución

El <recuadro> determina el tamaño según el valor:

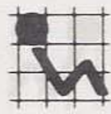
0:	8 x 8	sin ampliar
1:	8 x 8	ampliado
2:	16 x 16	sin ampliar
3:	16 x 16	ampliado

Ejemplo:

```
10 SCREEN 0, 1
20 LOCATE 10, 10 : PRINT "BEETHOVEN"
30 FOR I = 1 TO 500
40 NEXT
50 SCREEN 1.1
60 LOCATE 80, 80 : PRINT "BEETHOVEN"
70 FOR I = 1 TO 500
80 NEXT
90 SCREEN 2, 1
100 LOCATE 20, 60 : PRINT "BEETHOVEN"
110 GOTO 110
```

Este ejemplo demuestra los diferentes efectos conseguidos en los tres modos. Aunque los caracteres expuestos en pantalla mediante las líneas 20 y 60 son muy similares, observa las diferentes ubicaciones del cursor.

Cuando se cambia el modo de pantalla en la línea 90 a modo 2, los caracteres expuestos son mucho mayores.



4.2.1.2 WIDTH

Propósito: Establecer la anchura de pantalla durante el modo texto.

Versiones: Cassette, diskette.

Formato: WIDTH [<anchura de texto>]

Observaciones: La <anchura de texto> es una expresión numérica válida, cuyo valor ha de ser entero, bien 39, o bien 40. El valor prescrito para omisiones es 39.

Si se ha instalado el cartucho de 80 columnas y el monitor, también puede establecerse la anchura 80, cuando se opera con el BASIC en diskette.

La anchura corresponde al número de caracteres en cada línea de pantalla, y al cambiarla se hace que se limpie la pantalla.

Ejemplo:

```
10 FOR I = 1 TO 50
20 PRINT I;
30 NEXT
```

Ensaya el ejemplo anterior tecleando:
RUN

Luego teclea
WIDTH 40

y observas cómo se limpia la pantalla y cómo aparece la divisa Ok en la esquina superior izquierda de la pantalla. Teclea
RUN

para ejecutar el programa. Observa cómo ahora se expone un carácter más por línea con respecto a la anterior.

Si está instalado el cartucho de 80 columnas y el monitor está encendido, ensaya:
WIDTH 80

Ahora ejecuta de nuevo el programa anterior.



4.2.1.3 CLS

Propósito: Limpiar la pantalla.

Versiones: Cassette, diskette.

Formato: CLS

Observaciones: Borra la página activa en ese momento que se está mostrando en pantalla y vuelve el cursor a su posición base (la esquina superior izquierda de la pantalla). La instrucción SCREEN también fuerza a que se limpie la pantalla, si el modo de pantalla resultante de ejecutar la instrucción es diferente al modo corrientemente en vigor. Igualmente sucede con las instrucciones WIDTH.

La pantalla puede también limpiarse pulsando la tecla CLS o simultáneamente CTRL y L. Y también usando el comando PRINT CHR$(12).



4.2.1.4 LOCATE

Propósito: Situar el cursor en un lugar determinado de la pantalla.

Versiones: Cassette, diskette.

Formato: LOCATE <coordX>, <coordY> [,<llavecursor>]

Observaciones: <coordx> es cualquier expresión numérica entera, en la gama 1 a 40, o 1 a 80, dependiendo de la anchura de la pantalla; que determina la posición horizontal del cursor (la columna).

<coordy> es una expresión numérica entera en la gama 1 a 25, que indica la ubicación vertical del cursor (la fila).

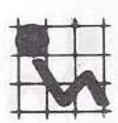
<llave cursor> es un valor que permite ocultar o exponer el cursor, según su valor sea 0 ó 1 respectivamente. Es válida en el modo texto.

Ejemplo:

```
10 CLS
20 LOCATE 5,5
30 PRINT "HOJA DE BALANCE"
40 LOCATE 5, 10, 1
50 PRINT "IMPORTE"
60 GOTO 60
```

Este ejemplo muestra los literales "HOJA DE BALANCE" e "IMPORTE" en dos líneas separadas.

En la línea 20 se ubica el cursor en la columna 5 de la fila 5. En la línea 40 se sitúa en la 5ª columna de la fila 10; y haciendo que el símbolo distintivo del cursor aparezca en la columna 1 de la fila 11, dado que la instrucción 50 no está terminada con punto y coma.



4.2.1.5 COLOR

Propósito: Fijar los colores prescritos para el fronte, el fondo y el reborde de la imagen.

Versiones: Cassette, diskette

Formato: COLOR [<frente>][,<fondo>] [,<reborde>]

Observaciones: En cada una de las 'mallas' de la cuadrícula que aparentemente configura la pantalla, se pueden estipular dos colores.

El color del fronte, que será el usado para mostrar los propios caracteres. El color del fondo que es el que rellenará el resto de la malla. Además, en la mayoría de los monitores de color y aparatos de TV, la imagen producida por el ordenador, está rodeada -rebordeada- por un área cuyo color también se puede prescribir con esta instrucción.

Los parámetros de la instrucción han de ser expresiones numéricas enteras en la gama 0 a 15. Los valores prescritos para omisiones son respectivamente 15,4,5. Los dieciséis posibles colores son:

0	transparente
1	negro
2	verde
3	verde pálido
4	azul oscuro
5	azul pálido
6	rojo oscuro
7	ciano
8	rojo medio
9	rojo pálido
10	amarillo oscuro
11	amarillo pálido
12	verde oscuro
13	magenta
14	gris
15	blanco



Ejemplo:

```
10 SCREEN 2
20 FOR I = 0 TO 7
30 CLS
40 COLOR I, I + 8
50 FOR T = 1 TO 5
60 LOCATE 10, 40 : PRINT "COLOR"
70 LOCATE 54, 80 : PRINT I "," I + 8
80 NEXT T, I
90 COLOR 15, 4, 5
```

Tanto los colores del fronte como del fondo cambian a medida que se ejecuta el programa, mostrando los respectivos números de color en pantalla.

La línea 40 estipula el color del fronte, según el valor de la variable I y el del fondo según el valor de I + 8. La línea 90 estipula que el color del fronte sea blanco (15) y que el color del fondo sea azul oscuro (4); y toda la plana en que aparece la imagen, estará rebordeada por un color azul pálido (5).



4.2.1.6 PUT SPRITE

Propósito: Estipular los atributos de los **sprites** de 'figuras' definidas en el comando SPRITE$(n).

Versiones: Cassette, diskette.

Formato: PUT SPRITE <numportor> [,<posición>] [,<color>] [,<numfigura>]

Nota: En la literatura sobre el tema, también se habla de 'bloques movibles', 'objetos en pantalla', y 'porta-imágenes'.

Observaciones: El <numportor> ha de ser un entero en la gama 0 a 31, e identifica a cada uno de los 32 posibles, especificando además su 'prioridad' para aparecer en pantalla. Cuando coinciden en una misma zona, el que está designado con número 0 oculta parcialmente al que está designado con número 1, el designado con 1 al de número 2, y así sucesivamente.

El <numfigura> especifica la 'figura' que el sprite ha de llevar al aparecer en pantalla, y corresponde al número mencionado en la instrucción SPRITE$(n). Ha de ser un número entero menor de 32 cuando el tamaño es 8 x 8, o menor de 16 cuando el tamaño es de 16 x 16. Si se omite, se adopta como valor el que esté reflejado en <numportor>.

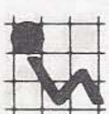
El parámetro <posición> puede mencionarse en la instrucción de dos maneras diferentes:

STEP (delta X, delta Y)
o (coordabs X, coordabs Y)

La primera forma hace referencia al incremento o decremento de las coordenadas **relativo** a las coordenadas más recientemente utilizadas para pintar en pantalla.

La segunda de las formas es la más habitual y hace referencia directamente a las coordenadas **absolutas** del punto de la pantalla. Ejemplos son:

(10, 10)	Coordenadas absolutas
STEP (10,0)	Incremento de 10 en la x y de 0 en la y.
(0, 0)	Coordenadas absolutas del origen.



Observa que cuando BASIC examina los valores de las coordenadas permitirá que pasen del borde de la pantalla, siempre que los valores caigan dentro de la gama de enteros (-32768 a +32767) en cuyo caso produciría un error de 'sobrepasamiento'. Todos los valores fuera del borde de la pantalla serán sustituidos con el valor legítimo más próximo, por ejemplo 0 para cualquier coordenada con valor negativo.

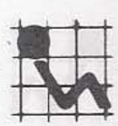
Observa también que las coordenadas (0,0) corresponden al punto origen de coordenadas, que es la esquina superior izquierda de la pantalla. Aunque parezca extraño, se comienza a numerar las coordenadas y a partir de la línea superior, así que la esquina inferior izquierda tendrá como coordenadas (0,191) en el modo de alta resolución, y ése es el standard.

Por tanto, la coordenada absoluta X ha de estar en la gama de -32 a 255; y la coordenada absoluta Y dentro de la gama -32 a +191.

Si a la coordenada y se le da el valor 208 (&HD0), en una determinada instrucción PUT SPRITE, todos los que tengan menor prioridad (<numportor> más alto) que el mencionado en dicha instrucción, desaparecerán de la pantalla y no volverán a aparecer hasta que se le vuelva a mencionar con un valor de y distinto de 208. Si el valor de y en la instrucción es 209 (&HD1), es el correspondiente a esa instrucción el que desaparece de la pantalla.

Consulta el manual VDP (Procesador de Imágenes de Video) para mayores detalles; pero recuerda que para hacer desaparecer un determinado <numportor> , debes fijar la coordenada y al valor 209; y para borrar todos los que tienen prioridad menor que uno dado, debes fijar el valor de la coordenada y a 208.

Cuando se omite el valor de un campo, se usa el valor que tenga prescrito en ese momento. Al inicio, si omites valor para el color, se adoptará el color corrientemente prescrito para el fondo.



Ejemplo:

```
10  FOR I = 1 TO 8
20  READ A$
30  B$ =B$ + CHR$ (VAL ("&B" + A$))
40  NEXT
50  SCREEN 1, 1
60  SPRITE$ (0) = B$
70  PUT SPRITE 1, (128, 96), 15, 0
80  GOTO 80
90  DATA 0
100 DATA 00111100
110 DATA 00100000
120 DATA 00111100
130 DATA 00000100
140 DATA 00000100
150 DATA 00111100
160 DATA 0
```

Lo que verás en la pantalla es una figura en forma de S y en color blanco. Dicha figura tiene especificada la forma en las líneas 90 a 160. En cada instrucción DATA hay 8 cifras binarias. Los ceros hacen que la mota correspondiente sea transparente, con lo que en pantalla aparecerá el color del fondo (o de la otra figura que coincida en posición) mientras que el 1 son motas que aparecerán en el color especificado en la instrucción 70. Las líneas 10 a 40 forman un bucle que permite apuntar los datos en la variable B$. Es en la línea 30 donde van obteniéndose los correspondientes números equivalentes, y posteriormente los caracteres, para formar un determinado literal en B$.

La línea 60 simplemente designa esa figura con el número 0.

Es la línea 70 la que hace que el 'sprite' número 1, lleve a pantalla en la posición (128,96) y con el color 15, la figura designada con 0.

Los recuadros ocupados por una figura definida por SPRITE$(n) no están limitados a las dimensiones 8 x 8. También pueden hacerse que sean de 16 x 16. Cuando se menciona en la instrucción SCREEN como <recuadro> el 0 o el 1, se elige un tamaño de 8 x 8; mientras que si se elige un valor 2 ó 3, el tamaño es de 16 x 16. El ejemplo siguiente muestra las diferencias.

```
10  SCREEN 1, 3
20  FOR X = 1 TO 32
30  READ A$
40  RESTORE
50  S$ = S$ + CHR$ (VAL (&B" + A$))
60  SPRITE $(0) = S$
70  PUT SPRITE 0, (128, 96), 15, 0
80  NEXT
90  GOTO 90
100 DATA 11110001
```

Observa que en el recuadro se rellena primeramente el rectángulo izquierdo de dimensiones 8 x 16, y luego el rectángulo contiguo de la derecha también con dimensiones 8 x 16.



4.2.1.7 CIRCLE

Propósito: Dibujar una **curva** -que es un arco de circunferencia o de elipse- con un centro y radio determinado.

Versiones: Cassette, diskette.

Formato: CIRCLE<centro>, <radio> [,<color>] [,<angcomienzo>] [,<angterminal>][,<curvatura>]

Observaciones: El <centro> viene dado por cualquiera de las dos formas de expresar coordenadas -relativas y absolutas- explicadas en la instrucción PUT SPRITE.

El <radio> fija el de la curva, y es un parámetro obligatorio.

El <color> especifica el usado para trazar la curva, y si se omite se hará en el color corrientemente prescrito para el frente.

Los parámetros <angcomienzo> y <angtérmino> son los ángulos, dados en radianes entre 0 y 2π, contados a partir del semi-eje horizontal positivo y hacia la izquierda que especifican el comienzo y el témino de la curva. Si sus valores son negativos, el arco quedará conectado al centro mediante una línea, y los ángulos serán considerados como si hubieran sido positivos. (Observa que eso es diferente de lo que se obtiene al añadir 2π).



<curvatura> corresponde a lo que en una elipse seríala relación de la altura y la anchura. El valor prescrito para omisiones es 1, con lo que la elipse es una circunferencia, y se supone que en la pantalla se emplea una cuadrícula cuya relación de la dos es 4/3. Si <curvatura> es menor que 1, el valor de <radio> se considera como puntos verticales; si la curvatura es mayor que 1 se consideran puntos horizontales.

Ejemplo:

```
10 SCREEN 1
20 CIRCLE (128, 96), 80, 15
30 GOTO 30
```

Aparecerá una circunferencia en blanco, centrada en (128, 96) con un radio de 80 puntos.

Ahora cambia la línea 20 para que sea:

```
20 CIRCLE (128, 96), 80, 15, 0, 3.14
```

y verás que sólo aparece la semi-circunferencia superior.

Cambia la línea 20 para que sea:

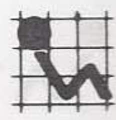
```
20 CIRCLE (128, 96), 80, 15,,,2
```

Y luego para que sea:

```
20 CIRCLE (128, 96), 80, 15,,,.5
```

Verás que en ambos casos se dibujan elipses, cuya relación entre altura y anchura depende del valor dado a <curvatura> .

Las tres comas que aparecen después del valor para <color>, son necesarias para indicar a la máquina que los ángulos de comienzo y término de la curva no se especifican, por lo que adoptará los valores prescritos para omisiones, que en este caso son 0 y 2π, respectivamente.



4.2.1.8 DRAW

Propósito: TRAZAR figuras lineales de acuerdo con un determinado lenguaje gráfico con **macro-instrucciones.**

Versiones: Cassette, diskette.

Formato: DRAW "<macro-instrucción gráfica>"

Observaciones: La figura a trazar se define mediante una serie de **macro-instrucciones** para gráficos, y se encierra entre comillas como cualquier constante literal; pero cuando se ejecute el comando DRAW será interpretada de acuerdo con las reglas para dicho lenguaje de gráficos. En ellas se establece mediante una 'sola letra' el trazo que ha de aparecer en pantalla.

Son las que numeramos a continuación, y hacen mover el 'pincel gráfico' a partir de la última posición que haya ocupado, y después de ejecutar cada una, la posición en que quede se toma como nueva posición para comenzar el trazo siguiente.

U <n>	Vertical arriba
D <n>	Vertical abajo
L <n>	Horizontal izquierda
R <n>	Horizontal derecha
E <n>	Diagonal arriba y derecha
F <n>	Diagonal abajo y derecha
G <n>	Diagonal abajo e izquierda
H <n>	Diagonal arriba e izquierda.

El argumento <n> en cada uno de los comandos anteriores, indica la distancia a recorrer. El número de **puntos** que se desplaza el 'pincel' es de n veces el factor de escala (estipulado con el comando S).

M <x>,<y> Mueve el pincel hasta un punto de coordenadas absolutas o relativas determinadas. Si x e y tienen un signo más (+) o un signo menos (-) precediendo a su valor, se consideran coordenadas relativas. Si no lo tienen, absolutas.

La **malla** de la cuadrícula aparente de la pantalla es 'cuadrada', y decimos que la 'relación de lados' es 1. Por lo tanto, en longitud son iguales los 8 puntos horizontales que los 8 puntos verticales. Se dice también que la **cuadratura** de la malla es 1.



Hay dos comandos que pueden preceder a cualquiera de las mencionadas como movimiento:

B	Desplaza el pincel, pero no pinta el trazo correspondiente.
N	Desplaza el pincel, pintando, pero vuelve a la posición original cuando termina el trazo.

También se dispone de los siguientes comandos:

A <n>	Fija el ángulo de inclinación de la figura trazada (el 'rumbo'), y su valor puede ser de 0 a 3, siendo 0 Norte, 1 Oeste, 2 Sur y 3 Este.
C <n>	Estipula el color, que puede ser un número de 0 a 15.
S <n>	Fija la **escala,** mediante un número en la gama 0 a 255. La escala utilizada será dicho valor dividido por 4. Por ejemplo, si n=1, ese factor de escala será 1/4. El factor de escala multiplicado por la distancia a trazar dada en los comandos U, D, L, R, E, F, G, H y en las coordenadas del comando M dan la distancia real trazada. El valor prescrito para omisiones es 1, lo que significa que no se usa escala, es decir, lo mismo que S4.

X <variable literal>
Pasa a trazar la figura definida en una subserie de macrocomandos gráficos, definida por una variable literal.

En todos estos comandos para gráficos, los parámetros <n> , < x > , <y> , pueden venir dados directamente por el valor (v.g. 123) o mediante la expresión = **< variable>** ; siendo < variable> el nombre de una variable numérica cuyo valor se utilizará para trazar la figura. Se necesita incluir un punto y coma (;) cuando se den los parámetros de esta forma, y también siempre que se use el comando X. Si no se da este caso, el punto y coma es de uso opcional para separar comandos.



Dentro de la serie de macrocomandos para gráficos, los 'espacios en blanco' no tienen ningún efecto, y ayudan a la legibilidad. Por ejemplo, puedes usar variables en un comando para movimiento del pincel, de esta manera:

M+=X1;,=X2;

El comando X es una faceta muy aprovechable de la instrucción DRAW, porque te sirve para definir parte de una figura a trazar, aisladamente de la figura completa (por ejemplo, las piernas); y también es aprovechable cuando la serie de macrocomandos ocupa más de 255 caracteres.

Ejemplo:

```
10 COLOR, 1, 1 : SCREEN 1
20 DRAW "C10 BM 100, 70 E15 R30 G15
   L30 D30 R30 U30"
30 DRAW "S4C8 BM 130, 100 E15 U30"
40 GOTO 40
```

Se trazará un cubo en perspectiva con dos 'pinceles', uno con amarillo y el otro en rojo; comenzando en (100, 70) y (130, 100) respectivamente. El factor de escala S4 no necesita especificarse, dado que es el valor prescrito para omisiones. (Compara el efecto de la línea 20 y la línea 30).

Sustituye ahora las líneas 20 y 30 por la siguiente línea:

```
20 DRAW "C8 BM 100, 70 E15 R30 D30 G15
   L30 U30 R30 NE15 D30"
```

Recuerda suprimir la línea 30 antes de ejecutar el programa modificado, y observa el uso del comando M. Con este otro programa

```
10 COLOR 7, 1 : SCREEN 1
20 DRAW "S4BM 100, 100 E50 F50 G50 H50"
30 DRAW "S2BM + 25, 25 U50 R50 D50 L50"
40 GOTO 40
```

Al ejecutarlo, aparecerá el cuadrado pequeño trazado por la línea 30, inscrito en un rombo mayor que está trazado por la línea 20. Con este otro ejemplo

```
10 SCREEN 1 : COLOR 7, 1
20 DRAW "BM 100, 50 F30 L60 E30"
30 X=-40 : Y=40
40 DRAW "BM + = X;, = Y; R80 F30 L140
   E30"
50 GOTO 50
```

Aparecerá un triángulo y un trapecio, aislados uno de otro. La línea 40 traza el trapecio a partir de la malla situada 40 puntos a la izquierda y 40 puntos por debajo del más recientemente mencionado, i.e. (100, 50), como tienes que deducir si rastreas el movimiento del pincel en la línea 20.



4.2.1.9 LINE

Propósito: Trazar una RECTA entre dos puntos determinados por sus coordenadas. Para ver el detalle de cómo especificar las coordenadas, consúltese la instrucción PUT SPRITE.

Versiones: Cassette, diskette.

Formato: LINE [<posicioncomienzo>] -<posicionterminal> [,<color>] [, B | BF]

Observaciones: Si se omiten las coordenadas que especifican la posición de comienzo de la recta, se tomará como punto de partida el correspondiente al más recientemente mencionado, y si no ha habido ninguno, se toma (0,0). Las coordenadas que especifican la posición terminal de la recta, pueden escribirse en forma absoluta o en forma relativa, añadiendo el incremento o decremento de coordenadas a añadir a las del <posición comienzo> . Por ejemplo, la instrucción

```
LINE (100, 100) - STEP (20, -20)
```

produce el mismo efecto que la instrucción

```
LINE (100, 100) - (120, 80)
```

La opción "B" especifica que se dibuje un rectángulo que tenga como diagonal la recta mencionada en la instrucción. La opción "BF" especifica que además de pintar el rectángulo correspondiente a la opción "B" se 'rellene' con el mismo color con que se ha trazado dicho rectángulo.

Ejemplo:

```
10 SCREEN 1
20 LINE (72, 72) - (200, 168), 15, B
30 LINE (72, 72) - (136, 36)
40 LINE - (200, 72)
50 LINE - (72, 72)
60 LINE (120, 108) - (152, 168),, BF
70 GOTO 70
```

La línea 40 especifica una recta desde la última posición a la que se ha hecho referencia (136, 36); y que termine en la posición (200, 72). Lo mismo sucede con la recta de la línea 50.



4.2.1.10 PAINT

Propósito: RELLENAR el recinto delimitado por una determinada curva cerrada, con un determinado color.

Versiones: Cassette, diskette.

Formato: PAINT <posicionbase>, [,<color>]

Observaciones: Las coordenadas de < posición base> pueden venir dadas de las dos maneras descritas en la instrucción PUT SPRITE, y dependiendo de dónde esté situado, la instrucción PAINT rellenará el recinto interior de la curva que contiene esa posición, o el recinto externo cuando esa posición cae fuera de la curva. Los valores de <posición base> no pueden sacar la posición fuera de los límites establecidos para la pantalla.

<color> puede estar en la gama de 0 a 15, y será el usado para la 'mancha' de color con que se rellena la curva.

La instrucción PAINT puede rellenar cualquier figura delimitada por una curva cerrada, pero al tropezar con curvas muy complejas y llenas de recovecos (v.g. las rías gallegas, o la costa brava), se puede producir un error de 'escaso de memoria'. Si sucede, debes usar la instrucción CLEAR para borrar las variables y aumentar el tamaño de la memoria reservada.

El color con que se rellena debiera ser el mismo que el utilizado para trazar la curva.

Ejemplo:

```
10 SCREEN 1
20 COLOR 15, 4
30 LINE (50, 50) - (205, 141), 8
40 LINE (50, 141) - (205, 50), 8
50 CIRCLE (128, 86), 90, 8
60 PAINT (135, 125), 8
70 GOTO 70
```

La línea 60 manda que tome como posición base para rellenar la curva (que bien pudiera ser una recta) en la posición (135, 125) y usando como color el 8 (magenta) hasta que alcance los bordes del recinto delimitado por la curva. La imagen final será un círculo cruzado por dos rectas, y con sólo el sector inferior formado en el círculo relleno de color.



4.2.1.11 PSET PRESET

Propósito: Pintar/repintar una mota de un color dado en una determinada posición de la pantalla.

Versiones: Cassette, diskette.

Formato: PSET <posición> [,<color>]
PRESET <posición> [,<color>]

Observaciones: La <posición> viene dada por las coordenadas en una de las dos maneras -absoluta o relativa- descritas en la instrucción PUT SPRITE.

El <color> es un entero dentro de la gama 0 a 15, y designa el color con que aparecerá la mota.

Ambas instrucciones -PSET/PRESET- pintan una mota en la posición designada, y cuando se menciona en las instrucciones el parámetro <color> operan exactamente de la misma manera.

La diferencia entre las dos instrucciones se observa cuando no se especifica explícitamente el color de la mota. En la instrucción PSET, se toma como color para pintar una mota, el que esté prescrito en ese momento para el fronte; mientras que con la instrucción PRESET, cuando se omite el <color> se toma el color prescrito en ese momento para el fondo.

Si la posición mencionada por sus coordenadas queda fuera de los bordes de la pantalla, no se efectúa ninguna acción y se presenta el correspondiente error. Si el valor dado para el color es mayor de 15, el error presentado es 'cita ilegal de función'.

Ejemplo:

```
10 SCREEN 2
20 FOR Y = 60 TO 120
30 PSET (100, Y), 6
40 FOR I = 1 TO 20 : NEXT
50 PRESET (100, Y)
60 NEXT
```

La aparición y desaparición de la mota de color crea la impresión óptica de que se va moviendo de arriba a abajo a través de la pantalla. La línea 40 es simplemente un retardo de tiempo, que determina la velocidad del movimiento.



4.2.1.12 KEY

Propósito: Asignar a una de las 10 teclas funcionales una constante literal o un comando BASIC.

Versiones: Cassette, diskette.

Formato: KEY <numtecla>, <literal | comando>

Observaciones: Hay 5 teclas en la parte superior del teclado, que pueden ser definidas por el usuario para que al ser pulsadas envíen directamente una cadena de caracteres que constituya una constante literal o un comando BASIC a usar en un programa o en la operación del ordenador.

<numtecla> identifica la tecla a la que se asigna el literal o comando, y será un número entero en la gama 1 a 10. El <literal/comando> no puede tener más de 15 caracteres, y si se le asignan solamente se tendrán en cuenta los 15 primeros.

Para dejar sin efecto el literal -o comando- asignado a una determinada tecla, basta asignarle la cadena vacía ("").

Ejemplo: Para asignar el comando PRINT TIME$ seguido de |ENTER| a la tecla de función número 1, usa el comando:

```
KEY 1, "PRINT TIME$" + CHR$(13)
```

para especificar una determinada constante literal, (que por lo tanto, al no ser comando no incluye CHR$(13)) puedes hacerlo mediante:

```
A$ = "***TECLEA SI/NO"
KEY 2, A$
```



4.2.1.13 KEY LIST

Propósito: Mostrar en pantalla los literales/comandos asignados a cada tecla de función.

Versiones: Cassette, diskette.

Formato: KEY LIST

Observaciones: Con este comando, aparecen los valores asignados a las 10 teclas de función, cada uno con sus 15 caracteres como máximo. La posición que cada literal ocupa en la lista corresponde al número de la tecla de función. Observa que si hay caracteres de control en el literal asignado a la tecla de función, aparecen como espacios en blanco.

```
KEY LIST
color        auto
goto         list
run          color 15, 4, 5
cload"       cont
list         run
Ok
```

Al poner en marcha el ordenador, esos comandos son los que están asignados a las teclas de función.



4.2.1.14 ON KEY GOSUB

Propósito: Preparar un cepo para captar la pulsación de las teclas de función durante la ejecución de un programa, y desviarse a un determinado número de línea del programa, según la tecla pulsada.

Versiones: Cassette, diskette.

Formato: ON KEY GOSUB <lista de nurolins>

Observaciones: Si el primer número de línea distinto de 0, de una rutina de manejo de interrupciones es asignado en una instrucción ON KEY GOSUB, se efectuará una comprobación de si alguna de las teclas de función está pulsada, cada vez que BASIC ejecuta una instrucción. Si hay alguna pulsada, el BASIC desviará el curso del programa hacia la rutina que comience con el número de línea homólogo de la <lista de nurolins> .

Cuando se capta algo en el cepo, se ejecuta automáticamente un comando KEY(n) STOP que desmonta todos los cepos preparados en el programa, para inhibir sus posibles acciones. La instrucción RETURN con que termina las rutinas de tratamiento de interrupciones, automáticamente efectuará un comando KEY(n) ON, a no ser que explícitamente se haya incluido dentro de la rutina de tratamiento la instrucción KEY(n) OFF.

Los cepos preparados para captar determinados eventos, no actúan cuando BASIC no está ejecutando un programa. Cuando el evento captado en el cepo es un error (provocando el desvío mencionado en una instrucción ON ERROR) automáticamente se desmontan todos los cepos que haya en el programa (incluyento ERROR, STRIG, STOP, SPRITE, INTERVAL y KEY).



Ejemplo:

```
10  ON KEY GOSUB 50, 90, 80
20  KEY (1) ON : KEY (2) ON : KEY (3) ON
30  CLS
40  C = 1
50  COLOR C
60  PRINT "*";
70  GOTO 60
80  BEEP
90  C = C + 1
100 IF C = 16 THEN C = 1
110 RETURN 50
```

Cuando se está ejecutando este programa, pulsa F1 y observarás que el color del asterisco permanece igual que si no lo hubieras pulsado. Pulsa en cambio F2 y verás que el color del asterisco cambia, ya que inmediatamente se ha ejecutado la subrutina que comienza en la línea 90. Si pulsas F3, también cambiará el color y además se oirá un pitido como consecuencia del desvío efectuado a la rutina que comienza en la línea 80.

Cambia la línea 20 de ese ejemplo para que sea:

```
20  KEY(1) OFF : KEY(2) OFF : KEY(3) OFF
```

y observa ahora que el color del asterisco no cambia cuando pulsas F2 o F3, ya que el cepo preparado en la línea 10, queda desmontado como consecuencia de las instrucciones de la línea 20.



4.2.1.15 KEY ON/OFF/STOP

Propósito: Permitir/reprimir la posible activación del cepo preparado con la instrucción ON KEY GOSUB..., o incluso 'desmontarlo' temporalmente.

Versiones: Cassette, diskette.

Formato: KEY (<numtecla>) ON | OFF | STOP

Observaciones: <numtecla> es el número identificativo de la tecla de función, y ha de ser entero en la gama 1 a 10. Una instrucción KEY(n) ON debe siempre ejecutarse para permitir que el cepo preparado para esa tecla de función, pueda ser 'disparado' al ser pulsada esa tecla. Después de ejecutar la instrucción KEY(n) ON, el BASIC cada vez que comienza una nueva instrucción, comprobará si la tecla mencionada está pulsada. Si lo está, desviará el cursor de la ejecución hacia el número de línea correspondiente que se haya especificado en la instrucción ON KEY GOSUB que previamente se le ha debido dar.

Si se ejecuta una instrucción KEY(n) OFF, no se activa el cepo preparado, aunque se pulse la tecla correspondiente; aunque sí se recuerda que ha tenido lugar dicha pulsación.

Si se ejecuta la instrucción KEY(n) STOP, se 'desmonta' el cepo preparado en la instrucción ON KEY GOSUB...; pero si se pulsa la tecla mencionada el BASIC lo recuerda de manera que activará inmediatamente el cepo si recibe a continuación el comando KEY(n) ON.

Ninguna de estas instrucciones tienen ningún efecto sobre el literal/comando asignado a la tecla de función correspondiente mediante el comando KEY...

Ejemplo: Consulta la instrucción ON KEY GOSUB.



4.2.1.16 ON STRIG GOSUB

Propósito: Preparar un cepo para que durante la ejecución del programa capte la posible opresión del botón de disparo del mando para juegos.

Versiones: Cassette, diskette.

Formato: ON STRIG GOSUB <lista de nurolins>

Observaciones: Cuando el cepo correspondiente se dispara, se ejecuta automáticamente una instrucción STRIG(n) STOP, de manera que quedan inhibidas las posibles activaciones de otros cepos preparados. La instrucción RETURN que señala el final de la rutina de tratamiento, automáticamente efectúa una instrucción STRIG(n) ON a no ser que explícitamente se le haya dado una instrucción STRIG(n) OFF dentro de la propia rutina suscitada por el cepo.

La captación (atrape) de estos eventos no tiene lugar cuando BASIC no está ejecutando un programa. Cuando se ha activado un cepo para errores, y se pasa a ejecutar la subrutina suscitada por la activación del cepo, se reprimen automáticamente todas las posibles activaciones de otros cepos preparados (incluyendo ERROR, STRIG, STOP, SPRITE, INTERVAL y KEY).

Ejemplo:

```
10 ON STRIG GOSUB 50, 60, 70
20 STRIG(0) ON : STRIG(1) ON : STRIG(2) ON
30 FOR T = 1 TO 500 : NEXT
40 GOTO 30
50 PRINT "BARRA ESPACIADORA" :
   STRIG(0) OFF : RETURN 30
60 PRINT "GATILLO DE MANGUETA I" :
   STRIG(1) OFF : RETURN 30
70 PRINT "GATILLO DE MANGUETA II" :
   STRIG(2) OFF : RETURN 30
```

Mediante la línea 10 se monta el triple cepo pertinente, designando los desvíos a hacer en cada caso; y es en la línea 20 donde se les permite que sean activados al pulsar los correspondientes mandos. Tomando como ejemplo la pulsación de la barra espaciadora, será ejecutada la primera vez que se pulse la rutina que comienza y termina en la línea 50; y como en ella explícitamente se reprimen las posteriores activaciones del cepo correspondiente, al pulsar por segunda vez la barra espaciadora, no será captado dicho evento. Igualmente ocurre para las líneas 60 y 70.



4.2.1.17 STRIG ON/OFF/STOP

Propósito: Reprimir la posible activación del cepo preparado mediante la instrucción ON STRIG GOSUB.

Versiones: Cassette, diskette.

Formato: STRIG (<numbotón>) ON | OFF | STOP

Observaciones: <numbotón> es un entero en la gama 0 a 2. Si su valor es 0, se usa como botón de disparo la barra espaciadora del teclado. Si es 1, corresponde al gatillo del joystick numerado como 1; y si es 2 la numerada como 2.

La instrucción STRIG(n) ON debe ejecutarse para permitir que pueda activarse el cepo preparado mediante la instrucción ON STRIG GOSUB, y el evento no queda recordado por el ordenador incluso aunque haya sucedido. Si se ejecuta la instrucción STRIG(n) OFF, se impide la posible activación del cepo y se suspende la comprobación del estado de <numbotón> durante el resto de la ejecución. Además, incluye aunque sea pulsado, no queda recuerdo de ese evento. Si se ejecuta la instrucción STRIG(n) STOP, también queda impedida la posible activación del cepo correspondiente, pero si se ha pulsado <numbotón> el evento es recordado por el ordenador, de manera que inmediatamente se activará el cepo cuando se ejecute una instrucción STRIG(n) ON.

Ejemplo: Consulta la instrucción ON STRIG GOSUB



4.2.1.18 ON STOP GOSUB

Propósito: Preparar un cepo para detectar la pulsación simultánea de las teclas CTRL y STOP, y designar un número de línea determinado cuando se produzca dicho evento.

Versiones: Cassette, diskette.

Formato: ON STOP GOSUB <nurolin>

Observaciones: Cuando se activa el cepo, automáticamente se ejecuta un STOP, de manera que se impide la posible activación posterior del cepo. La instrucción RETURN que señala el final de la rutina suscitada, automáticamente ejecutará una instrucción STOP ON, a no ser que explícitamente se le haya dado una instrucción STOP OFF dentro de la propia subrutina.

La detección de eventos, no tiene lugar cuando BASIC no está ejecutando un programa. Cuando se dispara el cepo de errores (preparado mediante la instrucción ON ERROR GOSUB), porque se ha producido dicho evento, automáticamente se inhiben las posibles activaciones de los otros cepos que haya preparados (incluyendo ERROR, STRIG, STOP, SPRITE, INTERVAL y KEY).

Ejemplo:

```
10 CLS
20 ON STOP GOSUB 70
30 STOP ON
40 PRINT "x";
50 GOTO 40
60 END
70 STOP OFF
80 PRINT : PRINT "PARADO CON CTRL
   Y STOP"
90 RETURN 60
```

Al pulsar durante la ejecución CTRL y STOP simultáneamente, se cede el control a la subrutina que comienza en la línea 70, debido al cepo preparado en la línea 20 y facultado para poder ser activado en la línea 30. Suprime las líneas 20, 30 y 60 e intenta ahora parar la ejecución pulsando CTRL y STOP. Observa las diferencias.

Prueba ahora con este otro ejemplo.

```
10 ON STOP GOSUB 40
20 STOP ON
30 GOTO 30
40 RETURN
```

Para pararlo, sólo queda apagar el ordenador.



4.2.1.19 STOP ON/OFF/STOP

Propósito: Facultar/reprimir la activación del cepo para CTRL-STOP, o desmontarlo.

Versiones: Cassette, diskette.

Formato: STOP ON/OFF/STOP

Observaciones: Debe ejecutarse una instrucción STOP ON para permitir que sea activado el cepo preparado para detectar la pulsación de CTRL-STOP. Después de ejecutar la instrucción STOP ON, si se ha especificado un número de línea en una instrucción ON STOP GOSUB, entonces cada vez que BASIC examine una nueva instrucción a ejecutar, comprobará si están pulsadas o no las teclas CTRL y STOP. Si lo están, hará que se desvíe el curso del programa al número de línea especificado en la instrucción ON STOP GOSUB.

Si se ejecuta una instrucción STOP OFF, se reprime la activación del cepo preparado para STOP, y no se guarda recuerdo del evento correspondiente aunque haya tenido lugar durante la ejecución del programa.

Sin embargo, si se ejecuta una instrucción STOP STOP, tampoco podrá ser activado el cepo correspondiente, pero sí queda constancia de que se ha producido el evento, de manera que se ejecutará inmediatamente la subrutina mencionada cuando posteriormente se ejecute una instrucción STOP ON.

Ejemplo: Consulta la instrucción ON STOP GOSUB.



4.2.1.20 ON SPRITE GOSUB

Propósito: Preparar un cepo para detectar la posible colisión entre figuras llevadas a pantalla mediante la instrucción PUT SPRITE, y hacer un desvío a un número de línea determinado, en caso de que se produzca dicho evento.

Versiones: Cassette, diskette.

Formato: ON SPRITE GOSUB <nurolin>

Observaciones: Cuando ocurra el 'choque' entre dos figuras, se ejecutará interna y automáticamente una instrucción SPRITE STOP, con lo que los posibles choques posteriores no serán detectados. La instrucción RETURN que señala la vuelta de la subrutina suscitada por el choque, hará automáticamente una instrucción SPRITE ON, a no ser que se haya incluido una instrucción SPRITE OFF dentro de la propia subrutina.

La detección de eventos de esta clase, no tiene lugar cuando BASIC no está ejecutando un programa. Cuando se activa un cepo para errores (preparado mediante la instrucción ON ERROR), automáticamente se inhiben todas las posibles activaciones de otros cepos (incluyendo ERROR, STRIG, STOP, SPRITE, INTERVAL y KEY).

Ejemplo:

```
10  ON SPRITE GOSUB 90
20  SCREEN 2, 2
30  A$ = " "
40  FOR I = 1 TO 32
50  READ B$
60  A$ = A$ + CHR$ (VAL ("&H" + B$))
70  NEXT
80  SPRITE $ (0) = A$
90  PUT SPRITE 0, (10, 10), 8, 0
100 PUT SPRITE 1, (110, 85), 11, 0
110 PUT SPRITE 2, (220, 170), 6, 0
120 GOTO 120
130 DATA 03, 0F, 1F, 39, 79, FF, FF, FF
140 DATA 2A, 2A, 2A, 4A, 4A, 52, 92, 92
150 DATA C0, F0, F8, 9C, 9E, FF, FF, FF
160 DATA 54, 54, 54, 52, 52, 4A, 49, 49
```




4.2.1.21 SPRITE ON/OFF/STOP

Propósito:	Facultar/impedir la activación del cepo, o incluso desmontarlo, preparado mediante la instrucción ON SPRITE GOSUB.
Versiones:	Cassette, diskette.
Formato:	SPRITE ON/OFF/STOP
Observaciones:	Debe ejecutarse una instrucción SPRITE ON para que pueda ser activado el cepo preparado mediante la instrucción ON SPRITE GOSUB para detectar los posibles choques. Después de la instrucción SPRITE ON, si está especificado un número de línea en la instrucción ON SPRITE GOSUB, cada vez que BASIC vaya a examinar una nueva instrucción a ejecutar, comprobará si las posiciones de las figuras mencionadas en las instrucciones PUT SPRITE coinciden. Si coinciden, automáticamente efectuará un desvío al número de línea especificado en la instrucción ON SPRITE GOSUB. Si se ejecuta una instrucción ON SPRITE OFF, se impide la activación del cepo correspondiente, y no queda registrado el evento aunque haya sucedido durante la ejecución. Si se ejecuta una instrucción SPRITE STOP, tampoco puede ser activado el cepo correspondiente, pero sí se guarda constancia de la posible colisión habida, de manera que inmediatamente que se ejecute una instrucción SPRITE ON se producirá el desvío a la subrutina pertinente.
Ejemplo:	Consulta la instrucción ON SPRITE GOSUB.



4.2.1.22 ON INTERVAL GOSUB

Propósito: Designar un número de línea al que se desviará el BASIC durante la ejecución de un programa, cada vez que haya transcurrido un determinado intervalo de tiempo.

Versiones: Cassette, diskette.

Formato: ON INTERVAL = <lapso> GOSUB <nurolin>

Observaciones: <lapso> fija el tiempo que ha de transcurrir entre dos desvíos sucesivos a <nurolin> y viene dado en segundos.

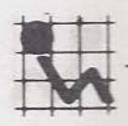
Cada vez que se detecta que ha transcurrido el tiempo especificado, se produce el desvío al número de línea correspondiente y automática e internamente se ejecuta una instrucción INTERVAL STOP de modo que las posibles activaciones del cepo posteriores no están permitidas. La instrucción RETURN que señala el final de la subrutina suscitada por haber transcurrido el tiempo, automáticamente efectúa una instrucción INTERVAL ON, a no ser que se haya incluido explícitamente la instrucción INTERVAL OFF dentro de la propia subrutina suscitada.

La detección de eventos de esta clase no se produce cuando BASIC no está ejecutando un programa. Cuando se activa un cepo para errores (preparado mediante la instrucción ON ERROR), automáticamente se impide la posible activación de otros cepos (incluyendo ERROR, STRIG, STOP, SPRITE, INTERVAL y KEY).

Ejemplo:

```
10  ON INTERVAL = 60 GOSUB 100
20  INTERVAL OFF
30  FOR I = 1 TO 100
40  PRINT I;
50  NEXT
60  INTERVAL ON
70  GOTO 70
100 BEEP : RETURN
```

Después de mostrar los números enteros del 1 al 100, se faculta la activación del cepo de 'crono' para que cada 1/60" emita un pitido, tal y como se especifica en la subrutina que comienza en la línea 100.



4.2.1.23 INTERVAL ON/OFF/STOP

Propósito: Habilitar/inhibir la detección de las señales de tiempo y la ejecución de las subrutinas mencionadas en la instrucción ON INTERVAL GOSUB.

Versiones: Cassette, diskette.

Formato: INTERVAL ON/OFF/STOP

Observaciones: Debe ejecutarse una instrucción INTERVAL ON para permitir que sea activado el 'cronómetro' puesto en marcha con la instrucción ON INTERVAL GOSUB. Después de ejecutar la instrucción INTERVAL ON, si hay mencionado un número de línea a la que producir el desvío, el BASIC cada vez que va a ejecutar una nueva instrucción comprueba si ha transcurrido o no el intervalo de tiempo especificado y efectúa cuando proceda un desvío al número de línea especificado en la instrucción ON INTERVAL GOSUB.

Si se ejecuta una instrucción INTERVAL OFF, no se produce la detección de los intervalos de tiempo, y aunque haya transcurrido el mencionado, no se ejecuta la subrutina ni tan siquiera queda recordado el evento en el sistema.

Si se ejecuta una instrucción INTERVAL STOP tampoco se produce el desvío a la subrutina correspondiente, pero si el tiempo había transcurrido, queda registrado así dentro del sistema, y se recuerda para que al ejecutarse posteriormente una instrucción INTERVAL ON, inmediatamente se produzca el desvío a la subrutina pertinente.

Ejemplo: Consulta la instrucción ON INTERVAL GOSUB.



4.2.1.24 VPOKE

Propósito: Inscribir en la memoria **visiva** del ordenador un determinado dato.

Versiones: Cassette, diskette.

Formato: VPOKE <exprnumdire>, <exprnumdato>

Observaciones: <exprnumdire> ha de ser un entero en la gama 0 a 16383 y expresa la dirección de la celdilla donde se va a grabar el dato. <exprnumdato> ha de ser un entero en la gama 0 a 255.

Ejemplo:

```
10  VPOKE &H3000, &H10
20  B = VPEEK (&H3000)
30  PRINT B
RUN
16
Ok
```

La línea 10 inscribe dentro de la celdilla de la memoria cuyos contenidos se exponen en pantalla -en la dirección 12288, el octeto &B00010000. La línea 20 examina el valor inscrito en esa dirección y en la línea 30 lo mostramos en notación decimal.

4.2.1.25 BEEP

Propósito: Emitir un pitido.

Versiones: Cassette, diskette.

Formato: BEEP

Observaciones: El pitido es el mismo que el conseguido mediante el comando PRINT CHR$(7).

Ejemplo:

```
10  FOR T = 1 TO 10
20  BEEP
30  NEXT
40  PRINT "*  *  *  *  *"
50  FOR T = 1 TO 10
60  PRINT CHR$(7)
70  NEXT
```

Emite 10 pitidos antes y después de exponer una ristra de 5 asteriscos. Son los comandos 20 y 60 los que le obligan a emitir el pitido.

4.2.1.26 MOTOR ON/OFF

Propósito: Conectar el motor de la lectograbadora de cassette.

Versiones: Cassette, diskette.

Formato: MOTOR [ON|OFF]

Observaciones: Cuando no se menciona ningún parámetro, simplemente 'bascula' el conmutador del motor, es decir, lo cierra si está abierto y lo abre si está cerrado.

Cuando se mencionan los argumentos, la palabra ON indica en marcha, y la palabra OFF quitado o en reposo.



4.2.1.27 SOUND

Propósito: Escribir directamente en los **registros** del generador programable de sonidos, los valores que fijan las cualidades del sonido.

Versiones: Cassette, diskette.

Formato: SOUND <registro del PSG>, <cualidad del sonido>

Observaciones: <registro del PSG> indica cuál de los 14 registros disponibles en el circuito generador programable de sonidos (PSG) es en el que debe 'registrarse' el valor. La <cualidad del sonido> es una expresión numérica entera en la gama 1 a 255, y depende del registro que se mencione en la instrucción, de acuerdo con la siguiente tabla:

REGISTRO	OPERACION	FUNCION
R0 - R5	Generador de tonos	Fija el período del tono en cada uno de los tres canales sonoros.
R6	Generador de ruido	Fija el período central de la banda de ruido.
R7	Mezclador	Expedita/impedita la emisión del sonido generado por cada uno de los canales, y sí/no le superpone el ruido prescrito para los tres.
R8 - R10	Control de amplitud	Especifica para cada uno de los tres canales el volumen del sonido emitido, a nivel permanente mientras dura el sonido o a nivel conformado según una determinada envolvente.
R11 - R13	Generador de envolvente	Controla la duración de la envolvente de volumen, y elige el perfil de dicha envolvente.



Control del Generador de Tonos

El PSG tiene 3 canales sonoros designados A, B y C. La frecuencia del sonido emitido por cada canal se obtiene a partir de la frecuencia base del reloj sincronizador, dividiéndola por 16 veces el valor de la frecuencia deseada:

<Valor deseado> = 3579545 / (16* frecuencia)

Registro bajo = <frecuencia deseada> AND 255

Registro alto = <frecuencia deseada> / 256

Los registros alto y bajo corresponden a la pareja de registros que controlan cada uno de los canales.

Canal	Registros (alto, bajo)
A	1,0
B	3,2
C	5,4

Ejemplo:

```
10 INPUT "TECLEA FRECUENCIA"; A
20 F = 3579545 / (16* A)
30 H = F / 256
40 L = F AND 255
50 SOUND 0, L
60 SOUND 1, H
70 SOUND 8, 15 : PRINT "CONTROLA VOLUMEN
   DE CANAL A"
80 SOUND 7, 254 : PRINT "&B 1111110 PARA
   ACTIVAR CANAL A"
```




Control del Mezclador

El circuito Mezclador combina el ruido con las notas emitidas por cada uno de los tres canales. La especificación de no combinar ninguno, o de combinar ruido y diferentes tonos en cada canal, se efectúa mediante la calibración binaria del registro número 7. Del bit 0 al bit 5 de dicho registro depende el canal y si se añade ruido o no. Los bits 6 y 7 son para los portales de acceso conectados a través del PSG y son ignorados por el BASIC. Para cada bit, el valor 1 desactiva el canal, mientras que el valor 0 lo activa.

Ejemplo:

SOUND 7, &B 11111110
Expide un tono por el canal A.

SOUND 7, &B 11110110
Expide ruido y sonido por el canal A.



Control de la Amplitud del Sonido

El PSG dispone de 3 registros independientes para controlar la amplitud del sonido emitido a través de los diferentes canales.

Canal	Registro
A	8
B	9
C	10

El valor a registrar para el nivel de volumen es de 0 a 15, siendo 15 el de mayor volumen.

Ejemplo:

```
10 SOUND 0, 100
20 SOUND 1, 0
30 SOUND 7, 254 : REM ACTIVA CANAL A
   (MEZCLADOR)
40 FOR I = 15 TO 9 STEP -1
50 SOUND 8, I
60 FOR J = 1 TO 200 : NEXT J : REM PAUSA
   SILENCIOSA
70 NEXT I
```

Este programa emite un sonido de tono alto, cuyo volumen se va desvaneciendo a causa del decremento especificado de 15 a 9.

El registro de control de amplitud también puede usarse para hacer que la amplitud dependa del perfil de envolvente establecido para cada canal; y para ello ha de colocarse en el correspondiente registro el valor 16 (&B00100000). En ese caso la amplitud del canal correspondiente estará controlada por los valores de los registros 11, 12, y 13. Consulta para más detalles el control de envolvente.



Valor	Perfil
0, 1, 2, 3, 9	
4, 5, 6, 7	
8	
10	
11	
12	
13	
14	
15	

Control de Período (duración) de la Envolvente

Los perfiles de envolvente pueden generarse de dos maneras diferentes: primeramente, se puede variar la frecuencia de la envolvente (y por tanto, la duración de la nota) por medio de los registros 11 y 12. En segundo lugar, se puede variar el perfil de la envolvente especificando los prescritos en el sistema, por medio del registro 13. Por ejemplo:

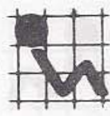
<frecuencia de envolvente deseada> = 3479545 / (256* frecuencia)

Ejemplo:

```
10  SOUND 0, 100
20  SOUND 1, 0 : REM TONO CANAL A
30  SOUND 7, &B 11111110 : REM EXPIDE A
40  SOUND 8, 16 : REM ACTIVA REG 11, 12
50  SOUND 13, 14 : REM ELIGE PERFIL
60  S= .5 : REM FREC = .5HZ
70  RELOJ = 3579545
80  L = RELOJ / (256* S) AND 255
90  H = RELOJ / (256* S) / 256
100 SOUND 11, L
110 SOUND 12, H
120 END
```

Control del perfil

Elige entre 9 diferentes perfiles de la envolvente de volumen del sonido emitido, programando el valor adecuado en el registro número 13.



PLAY "O4CN0N36"

L <n>
Fija la longitud (duración) de las notas que vienen a continuación. La longitud real de una nota es 1/n, siendo n un entero de 1 a 64. La siguiente tabla puede ayudar a explicar esto:

Longitud	Equivalencia
L1	Redonda o semibreve
L2	Blanca o mínima
L3	Una de un tresillo de tres blancas (1/3 en un compás de 4 tiempos)
L4	Negra o semínima
L5	Un quinto de un cinquillo (1/5 de compás)
L6	Una de un tresillo de negras
.	
.	
L8	Corchea
.	
L32	Fusa

El valor de la nota puede también ir a continuación cuando solamente quieres cambiar la duración de esa nota. Por ejemplo A16 es equivalente a L16A. El valor prescrito para omisiones es 4.

PLAY "CDEFGAB L16 CDEFGAB"

R <n>
Indica un silencio o reposo, y n es un entero en la gama 1 a 64; y se le aplican las mismas cifras comentadas para la L (longitud de la nota). También para omisiones se toma el 4.

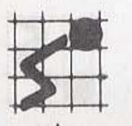
Se puede colocar un punto después de la nota haciendo que sea tocada como una nota 'sesquiada' (la longitud correspondiente se multiplica por 3/2). También puede colocarse más de un punto y la longitud se ajusta de acuerdo con el número de puntos. Por ejemplo, "A..." será tocada con una longitud 27/8 veces la prescrita. Igualmente se pueden colocar puntos después del valor dado para la pausa (R) y operará de la misma manera en cuanto a la duración de dicho reposo.



PSG DIAGRAMA DE BLOQUES

REGISTRO	BIT	B7	B6	B5	B4	B3	B2	B1	B0
R0	Canal A Período del Tono				8-BIT Ajuste fino A				
R1						4 BIT Ajuste aproximado A			
R2	Canal B Período del Tono				8-BIT Ajuste fino B				
R3						4 BIT Ajuste aproximado B			
R4	Canal C Período del Tono				8-BIT Ajuste fino C				
R5						4 BIT Ajuste aproximado C			
R6	Período del ruido				5-BIT Control del Período				
		IN /OUT			Ruido			Tono	
R7	Habilitar	10B	10A	C	B	A	C	B	A
R8	Canal A Amplitud				M	L3	L2	L1	L0
R9	Canal B Amplitud				M	L3	L2	L1	L0
R10	Canal C Amplitud				M	L3	L2	L1	L0
R11	Período de Envolvente				8-BIT Ajuste fino E				
R10					8-BIT Ajuste aproximado E				
R13	Perfil/Ciclo de Envolvente					CONT.	ATT.	ALT.	HOLD

LOS 14 REGISTROS DE CONTROL
(LECTURA/ESCRITURA)



4.2.1.28 PLAY

Propósito: Tocar música de acuerdo con las instrucciones dadas usando el lenguaje de macro-instrucciones para música.

Versiones: Cassette, diskette.

Formato: PLAY <exprlitA> [,<exprlitB> [,<exprlitC>]]

Observaciones: Las expresiones literales son series de macro-comandos de música, constituidas por un solo carácter. La instrucción PLAY se basa en un concepto similar al de la instrucción DRAW, al incrustar un "lenguaje musical de **macros**" en una constante literal. Cuando se especifica la cadena vacía, el canal correspondiente permanece silencioso.

Los comandos mono-carácter que se pueden usar en la instrucción PLAY, son:

De la A a la G con signos #, +, - opcionalmente. Hacen que toque la nota correspondiente a dicha letra en la octava corriente en ese momento. Un signo 'sostenido' (#) o un signo más (+) después de la letra, indica un agudo; y un signo menos (-) indica un grave. Los signos #, +, -, no están permitidos, a no ser que se correspondan con una de las teclas negras del piano. Por ejemplo, B# es una nota no válida.

PLAY "CDEFGAB"

O <n>
Octava. Estipula la octava a usar en las siguientes notas. Se dispone de 8 octavas, numeradas de 1 a 8. Y cada octava va de la C a la B (CDEFGAB). La octava 4 es la prescrita para omisiones.

PLAY "O71GCAFECDGCAB O5 CDC"

N <n>
Toca la nota correspondiente al número n, que debe ser un entero en la gama 0 a 96. n = 0 significa reposo o silencio. Esta es una manera alternativa de elegir las notas a tocar en lugar de especificar la octava (O n) y el nomre de la nota (A-G). Así, la C en la octava 4 tiene por número el 36.



Ejemplo: SOUND 13, 14

Genera un tono modulando arriba y abajo de acuerdo con el período de envolvente establecido en los registros 11 y 12, cuando esté activado el bit 4 del registro 8 (i.e. SOUND (8, 16) está estipulado).



```
PLAY "CDER2C..D..E.."
```

T <n>
Especifica el 'tempo' o número de compases con los que se tocará la melodía. El número n ha de ser entero y en la gama 32 a 255, e indica exactamente el número de negras por minuto. El valor prescrito para omisiones es 120.

```
PLAY "T32 CDEFGAB T255 GDEFGAB"
```

V <n>
Volumen. Fija el nivel de la nota, siendo n un entero de 0 a 15. Para omisiones se toma 8.

```
PLAY "VO CDEFGAB V15 CDEFGAB"
```

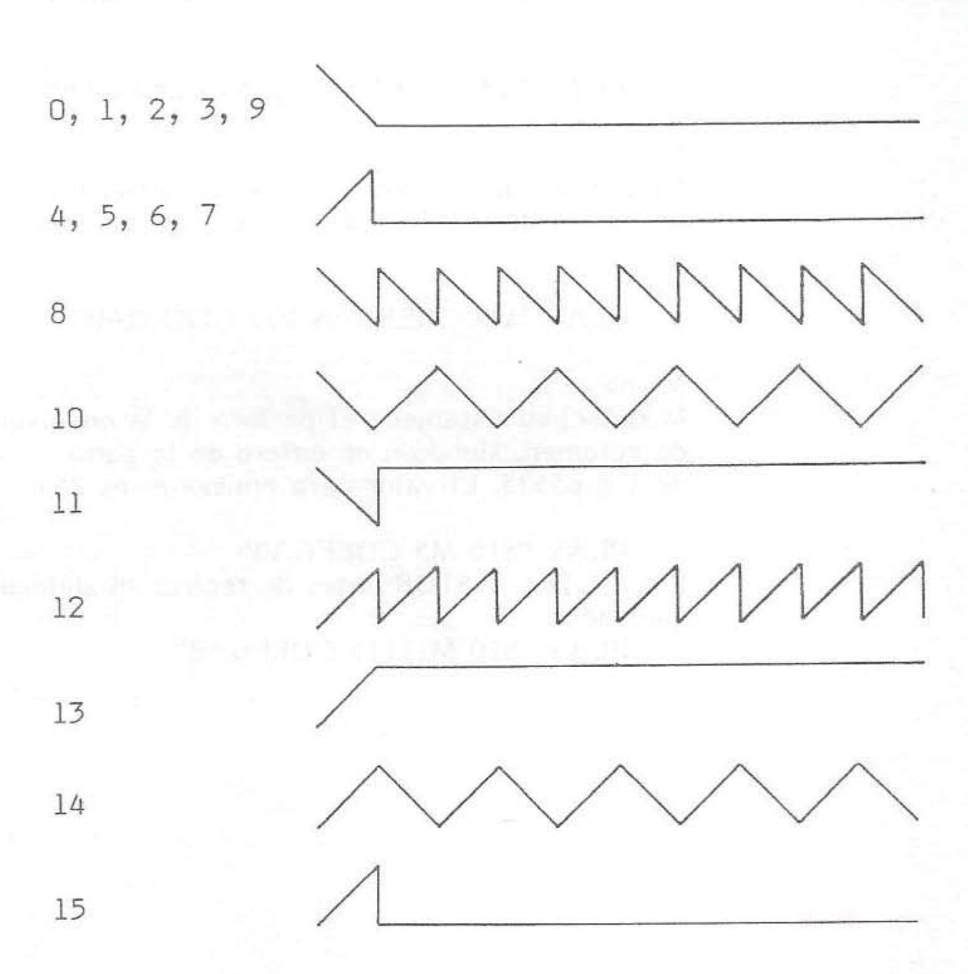
M <n>
Modulación. Establece el período de la envolvente de volumen, siendo n un entero en la gama de 1 a 65535. El valor para omisiones es 255.

```
PLAY "S10 M5 CDEFGAB"
```

Pulsa CTRL y STOP antes de teclear el siguiente comando:

```
PLAY "S10 M11115 CDEFGAB"
```

S <n>
La S corresponde al perfil de la envolvente, siendo n un entero en la gama de 1 a 15. El valor prescrito para omisiones es 1. El perfil o pauta de la envolvente corresponde con la siguiente tabla.




```
PLAY "S1 CDEFGAB"
PLAY "S15 CDEFGAB"
```




X <variable>;
Ejecuta la constante literal especificada.

```
10 A$ = "04FAAAEGGGDFEDC"
20 PLAY "04G05C040GECEGCGE05CCXA$;"
```

En todos estos comandos musicales, el argumento <n> puede venir dado por una constante como 12, o puede venir dado mediante la expresión = **<variable> ;** siendo <variable> el nombre de una variable literal. Se requiere el punto y coma (;) cuando se usa de esta manera una variable, y cuando se usa el comando X. Si no es así, el punto y coma es opcional para separar los comandos.

Observa que todos los valores especificados mediante los comandos mencionados anteriormente serán restaurados a los valores prescritos para omisiones, cuando se genera un pitido mediante BEEP.



4.2.1.29 MAXFILES

Propósito: Especificar el máximo número de ficheros que pueden estar abiertos simultáneamente.

Versiones: Cassette, diskette.

Formato: MAXFILES = <exprnum>

Observaciones: <exprnum> ha de ser un entero en la gama 0 a 15. Cuando se ejecuta MAXFILES=0 sólo pueden efectuarse SAVE y LOAD.

El valor prescrito para omisiones es 1.

Ejemplo:

```
10 MAXFILES = 3
20 OPEN "CAS : INDEX" FOR INPUT AS #1
30 OPEN "SCRN : CHAP 1" AS # 2
40 OPEN "KYBD : CHAP 2" AS # 3
.
.
.
.
.
```

La línea 10 especifica que el máximo número de ficheros que pueden estar abiertos en un momento dado es 3.



4.2.1.30 OPEN

Propósito: Reservar un área intermedia de memoria para entrada/salida, y especificar el modo que será usado con un fichero.

Versiones: Cassette, diskette.

Formato: OPEN "[<dispositivo>:] <nomefich>][FOR <modo>] AS [#] <nurofich>

Observaciones: Esta instrucción abre un 'cauce' de comunicaciones con un fichero en un determinado dispositivo, para el procesamiento posterior de los datos.

<modo> puede ser uno de los siguientes:

OUTPUT:	Modo secuencial en fichero de salida.
INPUT:	Modo secuencial en fichero de entrada.
APPEND:	Modo secuencial en fichero de salida, añadiéndole registros al final.

<nomefich> corresponde en la versión cassette, a una serie de 6 caracteres (máximo). Los nombres de los ficheros en discos pueden tener un máximo de 6 caracteres de longitud, con ampliación opcional del nombre que debe estar precedida por un punto decimal.

<nurofich> es una expresión entera cuyo valor ha de estar entre 1 y el máximo número de ficheros especificados en la instrucción MAXFILES. Es el número asociado al canal que comunica al ordenador con el equipo periférico correspondiente, y es el mencionado en las instrucciones de entrada/salida cuando se hace referencia a ese fichero.



Debe ejecutarse una instrucción OPEN antes de que pueda hacerse cualquier transferencia de entrada/salida al fichero, usando cualquiera de las siguientes instrucciones, o cualquiera de las funciones e instrucciones que exigen un número de fichero:

PRINT #, PRINT # USING
INPUT #, LINE INPUT #
INPUT$, GET, PUT

Cada fichero de datos está identificado por su nombre de fichero y por su número de fichero. El nombre del fichero es la etiqueta que usas para referirte al fichero. El número de fichero es lo que el ordenador usa para hacer referencia a ese fichero.

Ejemplo:

```
10  OPEN "1 : DATARIO" FOR OUTPUT AS # 1
20  A$ = "EMPLEADO"
30  B$ = "NOMBRE"
40  PRINT # 1, A$, B$
50  CLOSE # 1
60  C$ = "DEPARTAMENTO"
70  OPEN "1 : DATARIO" FOR APPEND AS # 1
80  PRINT # 1, C$
90  CLOSE # 1
100 OPEN "1 : DATARIO" FOR INPUT AS # 1
110 LINE INPUT # 1, D1$
120 LINE INPUT # 1, C1$
130 CLOSE # 1
```

La línea 10 instruye al ordenador para que abra o genere un fichero en la unidad ductora 1, denominado "DATARIO" sobre el que va a sacar o escribir información. "#1" al final de la línea 10 es el número de fichero asociado a partir de ese momento al fichero "DATARIO".

La línea 70-90 vuelve a abrir ese fichero, y luego leen de él la variable D1$ (que consta de A$ y B$) y la variable C1$ (que consta sólo de C$).



4.2.1.31 PRINT
PRINT # USING

Propósito: Enviar datos a un fichero mediante el canal especificado.

Versiones: Cassette, diskette.

Formato: PRINT # <nurofich>, <lista de datos>
PRINT # <nurofich>, USING <conformatriz>; <lista de datos>

Observaciones: Véanse las instrucciones PRINT y PRINT USING.

Ejemplo:

```
10  OPEN "1 : FICHAS" FOR OUTPUT AS # 1
20  A$ = "AMT1" = B$ = "AMT2" = C$ = "AMT3"
30  A = 12.235 : B = 64.2 : C = 129.653
40  PRINT # 1, A$, B$, C$
50  PRINT # 1, USING "$$###.##,"; A, B, C
60  CLOSE # 1
70  OPEN "1 : FICHAS" FOR INPUT AS # 1
80  LINE INPUT # 1, D$
90  CLOSE # 1
```

Las líneas 20 y 30 definen y asignan valor a las variables a utilizar. Las líneas 40 y 50 instruyen al ordenador para que las escriba en el diskette. El comando PRINT # 1... USING escribe datos numéricos en el disco sin usar delimitadores de campos explícitos. La coma al final del literal usado para conformar los datos, sirve para separar los valores numéricos grabados en el disco.



4.2.1.32 INPUT

Propósito: Recuperar datos de un fichero a través del cauce asociado e imponerlos como valores de las variables del programa.

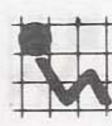
Versiones: Cassette, diskette.

Formato: INPUT # <nurofich>, <lista de variables>

Observaciones: <nurofich> es el número especificado cuando se abrió el fichero para entrada de datos.

<lista de variables> es la serie de variables que tendrán un dato de los tomados del fichero asignado a ellas. Pueden ser variables numéricas o litéricas, o elementos de una tabla. La clase de datos del fichero debe concordar con la clase de variables especificada en la lista. A diferencia de la instrucción INPUT, no se expone ningún signo de interrogación cuando se usa la instrucción INPUT # <nurofich> .

Los datos depositados en el fichero deben aparecer en él igual que si hubieran sido tecleados como respuesta a una instrucción INPUT. Con los valores numéricos, los espacios delanteros, marcas de avance de línea y retorno de carro se ignoran. El primer carácter que se encuentre que no sea un espacio en blanco o una marca de avance de línea o retorno de carro, se supone que es el primer carácter de un número. Se considera que termina el número cuando se encuentra un espacio en blanco, un avance de línea, o una coma.



Además, si el BASIC está examinando los datos en busca de uno literal, también se ignoran los espacios en blanco, y las marcas de avance de línea y retorno de carro. El primer carácter que se encuentre que no sea un espacio en blanco, es considerado como el arranque de un dato literal. Si el primer carácter son las comillas ("), el dato literal constará de todos los caracteres colocados entre esas primeras comillas y las segundas que aparezcan. Por lo tanto, un literal entrecomillado no puede contener dentro de él como carácter normal, otras comillas.

Si el primer carácter del literal no son las comillas, se dice que es un literal sin entrecomillar, y se considerará terminado cuando aparezca una coma, una marca de avance de línea o de retorno de carro, o después de haber tomado 255 caracteres. Si se alcanza la marca de final de fichero cuando se está ingresando un dato numérico o litérico, automáticamente se termina la introducción de ese dato.

Ejemplo:

```
10 OPEN "1 = DEMO" FOR OUTPUT AS # 1
20 A=10 : B=20 : C=30
30 PRINT # 1, A; B. C
40 CLOSE # 1
50 OPEN "1 : DEMO" FOR INPUT A$ # 1
60 INPUT # 1, A, B, C
70 CLOSE # 1
```

Este programa depositará los números 10, 20 y 30 en un fichero en disco y luego los recuperará del disco.

En la línea 50, se obliga al ordenador a volver a abrir el fichero; y observa que el número de fichero vuelve a ser 1.

La línea 60 es la que obliga al ordenador a recuperar los datos del fichero imponiéndolos como valores de las variables.



4.2.1.33 LINE INPUT#

Propósito: Recuperar una línea completa (máximo de 254 caracteres), de un fichero secuencial e imponerla como valor de una variable literal; sin considerar los posibles delimitadores.

Versiones: Cassette, diskette.

Formato: LINE INPUT# <nurofich>, <varlit>

Observaciones: <nurofich> es el número mencionado cuando se abrió el fichero, como cauce asociado al mismo.

<varlit> es el nombre de una variable literal a la que se le impondrá como valor la línea tomada del fichero.

LINE INPUT# recupera todos los caracteres que haya en el fichero secuencial hasta que se encuentre con una marca de retorno de carro. En ese momento, se salta completamente la secuencia de avance de línea-retorno de carro, y la siguiente instrucción LINE INPUT# leerá todos los caracteres hasta encontrar la siguiente marca. Si lo que se ha encontrado es una secuencia con las marcas avance de línea/retorno de carro, queda también conservada; es decir, dicho par de caracteres también son considerados como parte del literal impuesto como valor de la variable.

LINE INPUT# es especialmente aprovechable cuando cada línea del fichero ha sido compuesta por diferentes campos; o cuando el programa en BASIC ha sido guardado en el modo ASCII, y está siendo procesado por cualquier otro programa.



Ejemplo:

```
10 OPEN "1: DEMO" FOR OUTPUT AS # 1
20 A$A = "ESTO VA DE DEMO"
30 B$ = "APENDICE"
40 PRINT # 1, A$, B$
50 CLOSE "1: DEMO" FOR INPUT AS # 1
60 OPEN "1: DEMO" FOR INPUT AS # 1
70 LINE INPUT # 1, A$
80 CLOSE # 1
```

Este programa escribe el mensaje literal asignado a las variables en las líneas 20 y 30, en un fichero de salida, y posteriormente lo vuelve a tomar del mismo fichero abierto para entrada de datos.



4.2.1.34 INPUT$

Propósito: Ingresar una constante literal de un determinado número de caracteres, tomándola del teclado o de un fichero determinado.

Versiones: Cassette, diskette.

Formato: INPUT$ (<exprnum>, [#] <nurofich>

Observaciones: El argumento <n> de está función determina el número de caracteres a tomar del teclado o del fichero.

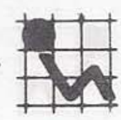
<nurofich> es el número de cauce asociado al fichero cuando fue abierto.

Si se usa el teclado para ingresar los caracteres, ninguno de ellos será mostrado en la pantalla. Todos los caracteres que se tecleen, incluyendo los caracteres de control pero exceptuando CTRL y STOP, constituirán el resultado literal de la función. Al responder a una función INPUT$ aplicada sobre el teclado, el resultado se consigue sin necesidad de pulsar la tecla ENTER. Para interrumpir la ejecución, se puede pulsar CTRL y STOP.

Ejemplo:

```
10 PRINT "DIME SI ES CORRECTA LA
   INSTRUCCION"
20 Z$ = INPUT$(1)
30 IF Z$ = "S" OR Z$ = "S" THEN PRINT
   "ESTAS SEGURO?"
40 IF Z$ = "N" OR Z$ = "n" THEN PRINT
   "!" ELSE PRINT "VAMOS, ACLARATE"
```

La línea 20 colecta el primer carácter impuesto mediante el teclado.



4.2.1.35 CLOSE

Propósito: Cerrar el canal asociado a un fichero, y dejar libre el espacio reservado en memoria usado para las transferencias de registros.

Versiones: Cassette, diskette.

Formato: CLOSE [[#] <nurofich> [,<nurofich>]]

Observaciones: <nurofich> es el número de cauce correspondiente al fichero que se mencionó en la instrucción OPEN. Cuando se ejecuta un comando CLOSE, se deshace la asociación establecida entre un fichero y un equipo de almacenamiento. Las operaciones posteriores de entrada/salida que especifiquen ese número de fichero no serán válidas; a no ser que vuelva a abrirse previamente el fichero. Una instrucción CLOSE que no mencione ningún número de fichero, hace que todos los que estén abiertos sean cerrados inmediatamente.

Ejemplo:

```
10 OPEN "1 : DEMO" FOR OUTPUT AS # 1
20 FOR I = 0 TO 50
30 PRINT # 1, I
40 NEXT I
50 CLOSE # 1
```

En la línea 50, se cierra el fichero que se abrió en la línea 10.



4.2.1.36 SAVE

Propósito: Guardar un programa en BASIC bajo la forma de fichero almacenado en un determinado equipo periférico.

Versiones: Cassette, diskette.

Formato: SAVE "[<dispositivo>:] <nomefich>" [,A]

Observaciones: El <dispositivo> puede ser: para cassettes, la palabra "CAS:" o simplemente omitirla. Para unidades ductoras de disco, deberá ser "1:" o "2:" dependiendo de si es la primera o segunda ductora de la unidad.

<nurofich> puede ser: para cassettes, un literal de 6 caracteres como máximo. Para diskettes, puede ser un literal con un máximo de 6 caracteres, seguido opcionalmente por una extensión del nombre precedida por un punto decimal. El número máximo de caracteres en esta ampliación es de 3.

Si el nombre del fichero tiene más de 6 caracteres, el BASIC inserta un punto decimal después del sexto, y usa automáticamente los siguientes 3 caracteres como ampliación del nombre. Cualquier punto decimal o carácter que venga a continuación es descartado.

Al guardar un fichero en cassette, se pone en marcha el motor y el fichero se graba en la cinta magnética.

Si ya existe un fichero con ese mismo nomefich será sobre-escrito.

La opción ",A" guarda el programa de acuerdo con los caracteres ASCII. Si no se elige esta opción, el BASIC guarda el fichero usando una notación binaria compacta ('taquizada'). Por ejemplo, los ficheros que intentaremos posteriormente congregar con el que haya en memoria (MERGE) deberán haber sido guardados con la opción ASCII.

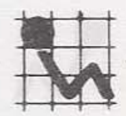
Ejemplo:

```
SAVE "CAROLINO"
```

Guarda en cinta el programa que hay en la memoria bajo el nombre de fichero "CAROLINO".

```
SAVE "1 : CURSOIPC.BAS"
```

El programa que haya en memoria, queda guardado en el diskette alojado en la unidad ductora 1.



4.2.1.37 LOAD

Propósito: Recuperar un programa en BASIC de un equipo de almacenamiento, y cargarlo en la memoria del ordenador.

Versiones: Cassette, diskette.

Formato: LOAD "[<dispositivo>:] [<nomefich>]" [,R]

Observaciones: El <dispositivo> puede ser: para cassette, la palabra "CAS:" o simplemente omitir el parámetro. Para diskettes, puede ser "1:" o "2:", dependiendo de la unidad donde se encuentre el diskette.

<nomefich> puede verse en la sección 3.13.1.2.

La instrucción LOAD automáticamente cierra todos los ficheros que pudiera haber abiertos y quita el programa que haya en la memoria. Sin embargo, si se usa la opción "R" todos los ficheros permanecen abiertos y además se ejecuta inmediatamente el programa cargado. Si se omite <nomefich> se carga el siguiente programa que haya en la cinta, y que debiera haber sido guardado como fichero ASCII.

Puede por tanto, usarse la instrucción LOAD con la opción R, para 'encadenar' varios programas, o segmentos de un mismo programa, traspasándose la información entre los programas usando los ficheros abiertos.

Ejemplo: LOAD "CURSOI3E"
Carga el programa mencionado, pero no lo ejecuta inmediatamente.

LOAD "CURSOI3E", R
Carga el programa mencionado, y automáticamente lo ejecuta.

4.2.1.38 MERGE

Propósito: Congregar las líneas de un programa guardado como fichero ASCII, con las líneas del programa que ocupa la memoria.

Versiones: Cassette, diskette.

Formato: MERGE "[<dispositivo>:] [<nomefich>]"

Observaciones: El <dispositivo> puede ser "CAS:", "1:", o "2:". Si se omite, se considera el cassette.

<nomefich> Consulta la sección 3.13.1.2.

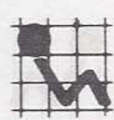
Si cualquiera de las líneas en el programa que se agrega al de memoria, coincide con las del programa que ya hay en la memoria, las líneas del fichero sustituirán las líneas de la memoria.

Después del comando MERGE, el programa 'mezclado' queda ocupando la memoria y el BASIC vuelve al nivel de comando.

Si se omite el <nomefich> , se agregará al de memoria el primer fichero que se encuentre; y como siempre que se mezclan, ha debido ser guardado como un fichero ASCII.

Ejemplo: MERGE "1: TUTORIAL"
Este comando mezcla el fichero llamado "TUTORIAL", que está depositado en el diskette de la unidad 1, con el programa que haya en memoria.

El programa "TUTORIAL" debiera haber sido guardado como un fichero ASCII. Los números de línea del programa que se toma del fichero en diskette, son predominantes sobre los posibles números de línea coincidentes que hubiera en el programa en memoria.



4.2.1.39 BSAVE

Propósito: Guardar una imagen de la memoria en un dispositivo de almacenamiento externo.

Versiones: Cassette, diskette.

Formato: BSAVE "[<dispositivo>:] <nomefich>", <direante>, <direulti> [,<direrule>]

Observaciones: El <dispositivo> puede ser uno de los siguientes: "CAS:", "1:" o "2:", o bien puede omitirse, en cuyo caso se considera el cassette.

Los parámetros <direante> y <direulti> son expresiones numéricas enteras que corresponden respectivamente a la dirección inicial (anterior) y a la dirección final (ulterior) de la sección de memoria cuyo contenido binario va a ser guardado en el dispositivo de almacenamiento.

Si se omite el parámetro <direrrule> se considera la dirección inicial como dirección en la que se ejecutará el programa una vez cargado, cuando el recinto de memoria corresponda a un programa en lenguaje máquina.

Ejemplo: BSAVE "CAS:" TEST", &HA000, &HAFFF

El contenido de la memoria desde la dirección 40960 hasta la dirección 45055, ambas inclusive, se guarda en cassette bajo el nombre de fichero "TEST".

4.2.1.40 BLOAD

Propósito: Cargar en memoria un fichero cuyo contenido fue previamente guardado mediante la instrucción BSAVE.

Versiones: Cassette, diskette.

Formato: BLOAD "[<dispositivo>:] [<nomefich>]" [,R] [,<incremento>]

Observaciones: El <dispositivo> puede ser uno de los siguientes: "CAS:", "1:", o "2:".

<nomefich> Consulta la sección 3.13.1.2.

Si se especifica la opción "R", después de ser cargado se ejecuta automáticamente el programa a partir de la dirección que se especificó en la instrucción BSAVE.

El contenido binario del fichero será cargado en la memoria a partir de la dirección que se mencionó en el comando BSAVE. Si se especifica en la instrucción el parámetro <incremento>, todas las direcciones que se hayan especificado en la instrucción BSAVE se verán incrementadas en ese valor.

Si se omite <nomefich> , se cargará el siguiente programa en lenguaje máquina que se encuentre en el equipo de almacenamiento.

Ejemplo: BLOAD "1 : SVFRMT", R

El programa cuyo nombre de fichero en diskette es "SVFRMT" será recuperado del diskette y cargado en la memoria, con ejecución automática una vez cargado.

4.2.1.41 CSAVE

Propósito: Guardar como un fichero en cassette el programa en BASIC que ocupa la memoria.

Versiones: Cassette.

Formato: CSAVE "<nomefich>" [,<velocidad en baudios>]

Observaciones: <nomefich> es el nombre con el que se identificará el programa en el cassette, y el número máximo de caracteres de dicho nombre es seis.

El BASIC guarda los programas en cinta según una notación binaria compacta ('taquizada'). Los ficheros con notación ASCII ocupan más espacio, pero en algunas ocasiones es preciso que los ficheros estén en dicha notación, porque van a ser procesados -usándolos como ficheros de datos- por otros programas. En esos casos, usa el comando SAVE.

Ejemplo: CSAVE "DEMO"

El programa existente en memoria se guarda en cassette bajo el nombre "DEMO".

4.2.1.42 CLOAD

Propósito: Recuperar de un cassette un programa en BASIC, y cargarlo en memoria.

Versiones: Cassette.

Formato: CLOAD "[<nomefich>]

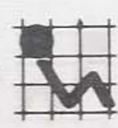
Observaciones: <nomefich> es un literal con 6 caracteres como máximo.

El comando CLOAD cierra todos los ficheros abiertos y quita el programa existente en memoria. Si se omite <nomefich> , se cargará el siguiente programa que se encuentre en la cinta. Para todas las operaciones de lectura de cassette, la velocidad en baudios está determinada automáticamente.

Ejemplo:

```
CLOAD "INTRO"
```

El fichero en cinta cuyo nombre es "INTRO" es trasvasado del cassette a la memoria del ordenador.



4.2.1.43 CLOAD?

Propósito: Verificar un programa en BASIC guardado en un cassette, con el programa existente en memoria.

Versiones: Cassette.

Formato: CLOAD? <nomefich>

Observaciones: <nomefich> es un literal de 6 caracteres como máximo.

Si el programa guardado en cinta no es exactamente igual que el programa que ocupa la memoria, se mostrará un mensaje de 'error en verificación' ("verify error").

Ejemplo: CLOAD? "CURSOI3E"

4.2.2.1 POINT

Propósito: Saber el color con que aparece un determinado punto de la pantalla.

Versiones: Cassette, diskette.

Formato: POINT (<coordX>, <coordY>)

Observaciones: Los argumentos de la función representan respectivamente la posición horizontal y la posición vertical del punto cuyo color se quiere examinar. Las coordenadas deben darse en valores absolutos.

Si el punto está fuera de la gama permitida, el resultado de la función es -1.

Ejemplo:

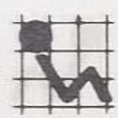
```
10 SCREEN 1
20 COLOR 15, 4, 5
30 PSET (200, 100), 8
40 A = POINT (200, 100)
50 B = POINT (100, 100)
60 C = POINT (200, 200)
70 SCREEN 0
80 PRINT A; B; C;
```

La línea 40 asigna a la variable A el color del punto (200, 100). Igualmente en las líneas 50 y 60 se asignan a las variables B y C los colores correspondientes a otros puntos. Al ejecutar el programa, el resultado obtenido es 8, 4 y -1.

El número de color 8 del punto (200, 100) se especificó en la línea 30.

El número de color 4 correspondiente al punto (100, 100) es el color del fondo que se determinó en la línea 20.

Como ya hemos dicho, cuando el punto cae fuera de la gama permitida, e.g. (200, 200), el resultado de la función es -1.



4.2.2.2 VPEEK

Propósito: Examinar el contenido de una determinada celdilla de la memoria visiva.

Versiones: Cassette, diskette.

Formato: VPEEK (<exprnumdire>)

Observaciones: El argumento <exprnumdire> ha de ser entero y en la gama de 0 a 16383; y el resultado de la función es un entero en la gama 0 a 255.

Consulta el comando VPOKE que es el complementario de esta función.

Ejemplo:

```
10  VPOKE &H3000, &H22
20  B = VPEEK (&H3000)
30  PRINT B
RUN
34
Ok
```

La línea 20 mira el valor contenido en la celdilla cuya dirección es el argumento de la función, que en este caso es 12288; y el valor es asignado a la variable B que al exponerla en pantalla da el número 34 (que en hexadecimal es el 22).



4.2.2.3 STICK

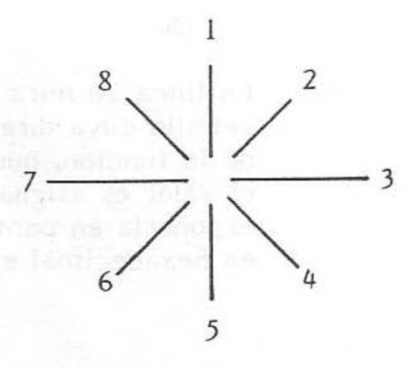
Propósito: Examinar la dirección del joystick de juegos.

Versiones: Cassette, diskette.

Formato: STICK (<exprnum>)

Observaciones: El argumento <exprnum> ha de ser un entero en la gama 0 a 2. Si es igual a 0, se usan las teclas de movimiento del cursor como joystick de juego. Si es 1 ó 2, corresponderá al primero o segundo de los joysticks para juegos conectados al correspondiente portal del ordenador.

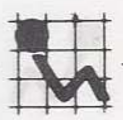
Si no está accionado el correspondiente bastón, el valor resultado de la función es 0. Si lo está, el valor corresponde a la dirección que señala el bastón, de acuerdo con el siguiente diagrama.



Ejemplo:

```
10  SCREEN 0:
20  X% = 20 : Y% = 12
30  LOCATE X%, Y% : PRINT "*"
40  S = STICK (0)
50  IF S = 0 OR S = 1 OR S = 5 THEN 40
60  LOCATE X%, Y% : PRINT " "
70  ON (S + 1)/4 GOTO 80, 100
80  X% = X% + 1 : IF X% = 39 THEN X% = 0
90  GOTO 30
100 X% = X% - 1 : IF X% = -1 THEN X% = 38
110 GOTO 30
```

Este programa muestra cómo se desplaza un asterisco a través de la pantalla. En la línea 40, la dirección señalada por las teclas del cursor queda registrada como valor de la variable S. La línea 50 restringe los desplazamientos del asterisco para que tengan por lo menos componente de movimiento hacia la izquierda o la derecha.



4.2.2.4 STRIG

Propósito: Examinar el estado del botón de disparo de un joystick para juegos.

Versiones: Cassette, diskette.

Formato: STRIG (<exprnum>)

Observaciones: El argumento <exprnum> ha de ser un entero en la gama 0 a 2. Si es 0, se considera como botón de disparo la barra espaciadora del teclado. Si es 1 ó 2, se examina el botón de disparo del correspondiente joystick para juegos.

Si no está pulsado el botón, el resultado de la función es 0. Si lo está, el resultado es -1.

Ejemplo:

```
10 CLS
20 COLOR 15
30 IF STRIG(0) THEN GOSUB 60
40 PRINT "*"
50 GOTO 30
60 PRINT "#"
70 RETURN 30
```

A medida que se ejecuta el programa, va apareciendo una figura formada por asteriscos "*" y diési "#".

Merced a la línea 30, al pulsar la barra espaciadora, se expondrá en pantalla el carácter # en lugar del carácter *.



4.2.2.5 PAD

Propósito: Examinar los diversos estados posibles en el tablero gráfico.

Versiones: Cassette, diskette.

Formato: PAD (<exprnum>)

Observaciones: El argumento <exprnum> es un entero en la gama de 0 a 3.

Cuando dicho argumento es 0, el resultado de la función es -1 cuando está accionado el palpador del tablero, y 0 cuando no lo está.

Cuando el argumento es igual a 1, el resultado corresponde a la coordenada x del tablero, que está en la gama 0 a 255.

Cuando el argumento es igual a 2, el resultado corresponde a la coordenada y del tablero, que está en la gama 0 a 255.

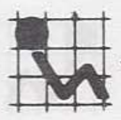
Cuando el argumento es igual a 3, el resultado corresponde al estado del pulsador en el lápiz palpador, siendo -1 si está pulsado y 0 en caso contrario.

Ejemplo:

```
10 SCREEN 0
20 COLOR 15, 5, 5
30 CLS
40 IF PAD (0) THEN 60
50 GOTO 40
60 X = PAD(1) : Y = PAD(2)
70 PSET (K, Y)
80 IF PAD(0) THEN 100
90 GOTO 80
100 X = PAD(1) : Y = PAD(2)
110 LINE - (x, y)
120 GOTO 80
```

Se mostrará en pantalla el diagrama geométrico dibujado en el tablero gráfico.

Las líneas 40 y 80 comprueban el estado correspondiente al tablero. Las líneas 60 y 100 imponen como valores de las variables x e y las coordenadas correspondientes del punto 'pinchado' en el tablero.



4.2.2.6 EOF

Propósito: Avisar de que el final de fichero ha sido alcanzado.

Versiones: Cassette, diskette.

Formato: EOF (<nurofich>)

Observaciones: El argumento <nurofich> es el número del cauce asociado al fichero y especificado en la instrucción OPEN.

El resultado es -1 (cierto) cuando se ha alcanzado el final de un fichero secuencial. En caso contrario es 0. Usa la función EOF para comprobar si ya has llegado a la marca de final de fichero cuando estás ingresando datos del mismo; para evitar que te aparezca el error 'final de fichero sobrepasado' ("Input past end").

Ejemplo:

```
10  OPEN "1 : DEMO" FOR OUTPUT AS # 1
20  FOR A = 0 TO 50
30  PRINT # 1, A
40  NEXT A
50  CLOSE # 1
60  OPEN "1 : DEMO" FOR INPUT AS # 2
70  IF EOF(1) THEN GOTO 110
80  INPUT # 1, A
90  GOTO 70
100 CLOSE # 1
110 END
```

Este programa deposita en un fichero los números del 0 al 50, y luego recupera esos datos del mismo fichero.

En la línea 70, se emplea la función EOF para comprobar si se ha alcanzado o no el final del fichero. Si el resultado de la función es cierto, se da un salto en el transcurso del programa a la línea 110 para terminarlo, si no se prosigue ingresando datos.



4.2.2.7 PLAY

Propósito: Examinar el estado de un canal sonoro.

Versiones: Cassette, diskette.

Formato: PLAY (<canal de sonido>)

Observaciones: Esta función indica por su resultado si en el canal de sonido hay o no un 'chorro' de notas pendientes de ser emitidas.

El argumento <canal de sonido> ha de ser un entero en la gama de 0 a 3. Si es 0, se efectúa una operación OR con los estados de los tres canales y se entrega el resultado de dicha operación.

Si el valor es 1, 2 ó 3, el resultado de la función corresponde al estado del canal pertinente, y será -1 si todavía está el canal en operación (le quedan notas por sonar); y 0 en caso contrario.



```
10      ONSTOP GOSUB 410: STOP ON
20      CLS
30      COLOR 15, 2, 2
40      SCREEN 1
50      LOCATE 5,  88:  PRINT " *****************************************"
60      LOCATE 5,  96:  PRINT " *SPECTRAVIDEO ADSR, 3 CHANNEL MUSIC DEMO *"
70      LOCATE 5, 104:  PRINT " *****************************************"
80      COLOR 15, 1, 1
90      PLAY "t60l16s0m8963o5:, "t60l16V10m8963o3", "t60l16V10o3m8963"
100     PLAY "d-4.ob-g-e-89-b-05d-8ob-g-e-8g-b-g-8g-e-", "r1r4.b", "r1re.g-"
110     PLAY "g-8fe-m26890d-1m8963e-89-8", "03b-2", "g-2"
120     COLOR 15, 10, 10
130     PLAY "a-4.o5d-8ob-8g-b-a-8o5d-8ob-4o5", "of2g-4f2:, "d-2e-8d-4.d4"
140     PLAY "e-2ob-4.b805d-4.ob-g-e-8g-b-", "b-2b-4.b8od-2e-4", "e-2"
150     PLAY "o5d-8ob-g-e-8g-b-g-8g-e-", "d-4e-4d-8o3b8", "o3b-40c4o3b-8a-8"
160     PLAY "g-8fe-d-1r803", "b-2.b-2", "g-eod-4d-4d-o3bb-a-g-8fe-d-4de-o2b"
170     PLAY "b-od-e-8g-a-18bo5d-e-g-4116", "r8od-2.b-4", "b-a-03b2.o9-4"
180     PLAY "fe-d-4V5d-s0obb-a-", "b-4", "g-4o3"
190     PLAY "g-a-b-fe-a-d-o3bof", "e-8.fe-8.d-o3b8", "b8.od-o3b8.b-a-8"
200     PLAY "e-o3b-a-od-o3c-8.od-f-a-bo5d-f-a-", "bb-a-2.", "a-g-f-2"
210     PLAY "b8b-a-g-2b-8a-g-e-4.d-ob", "o5e-2.og-4o3b2", "oblf2"
220     PLAY "b-8a-g-e-4.e-c", "b4b-2", "g-4o2b-8r8b-4"
230     COLOR 15, 13, 13
240     PLAY "o2b-8b-o3ce-fgb-oc8e-c", "oe-2o3a-4", "o39g2f4"
250     PLAY "e-8o3b-oce-fgb-o5c8e-c", "o2b-8.o3ce-fgb-oc8e-c", "g20f4"
260     PLAY "e-8ob-o5ce-fg-b-o6e-4.", "98o3b-oce-fg-b-b4.", "o4b-2o5b4."
270     PLAY "124d-e-d-o5b-8a-8116V5a-s09-fe-124", "o5a-8g-8b4.o", "f8e-2o"
280     PLAY "e-8d-ed-116ob-8a-4g-a-", "b8a-8g-8a-4o39-of", "g-8f8e-8c4b8"
290     PLAY "18b-.g-16e-g-b-o5", "18e-.d-16o3b-od-e-", "18o3b-.g-16e-g-b-o"
300     PLAY "d-ob-g-e-4g-4b-.", "g-e-d-o3b-4b4oe-.", "d-o3b-g-e-4g-4b-."
310     PLAY "g-16b-o5d-e-g-b-g-", "d-16e-g-b-o5d-e-d-", "g-16b-od-e-g-b-g-"
320     PLAY "ob-4d-4r2o6d-4v4d-4.s016", "o3c-ea-4og-1g-", "o39-4f4oe-1e-"
330     PLAY "o5b-g-e-8g-b-o6d-8o5b-g-e-8g-b-g-8g-e-", "g-g-1", "e-e-1"
340     PLAY "1g-fe-d.o18bb-a-116", "b-2.d-2.116", "g-1.116o3"
350     PLAY "g-a-b-fe-a-g-d-o3bof", "e-8.fe-8.d-o3b8", "b8.od-o3b8.b-a-8"
360     PLAY "e-o3b-a-od-e-o3b-bo", "bb-a-8.b-b", "a-g-f8.9-a-"
370     PLAY "d-e-o3b-a-o", "b8b-a-", "a-8g-f-"
380     PLAY "d-e-o3b-bog-a-bo5d-e-g-a-", "a-8oe-8d-e-g-a-bo5d-e-", "f8g-a-2"
390     PLAY "L12B06D-E-0", "L12G-A-BB", "A-4A-24"
400     PLAY "s0m53780o9-1.", "s0o6r16d-1.", "s0o5r32b-1."
410     COLOR 15, 4, 5
```



4.2.3 Variables especiales

Las siguientes, son variables especiales. Cuando colocadas en la parte derecha de una instrucción de asignación, son evaluadas como una función y su valor actual se entrega como resultado de la función.

Cuando se usan como variables, colocándolos en la parte izquierda de una instrucción de asignación, se comportan como cualquier otra de las variables definibles por el usuario.



4.2.3.1 TIME

Propósito: Cronómetro del sistema (temporizador).

Versiones: Cassette, diskette.

Formato: TIME

Observaciones: Corresponde a una variable numérica entera sin signo, y automáticamente se incrementa en una unidad cada vez que el procesador de imágenes de video genera una señal de interrupción (60 veces por segundo); y por lo tanto, cuando dicha señal de interrupción está inhibida (por ejemplo, al manipular el cassette) conserva el último valor que tenía.

Ejemplo:

```
10 DEFINT H-S
20 TIME = 0
30 T = TIME/60
40 H = T/3600
50 M = (T - 3600*M)/60
60 S = T - 60*M - 3600*M
70 SCREEN 2
80 PRINT H":" M":" S
90 PRINT TIME
100 GOTO 30
```

Este programa sirve como reloj.

Observa que el valor de la variable especial TIME se incrementa en 60 cuando la variable S se incrementa en 1.



4.2.3.2 SPRITE$

Propósito: Examinar o asignar las figuras que pueden ser llevadas por sprites.

Versiones: Cassette, diskette.

Formato: SPRITE$ (<número de figura>)

Observaciones: El argumento <número de figura> debe ser entero y menor de 256 cuando el tamaño estipulado es 0 ó 1; y entero y menor de 64 cuando el tamaño es 2 ó 3. (Pero recuerda que solamente puede haber en pantalla 32 sprites).

La figura viene definida por un dato literal o alfanumérico cuya longitud ha de ser ineludiblemente de 8 caracteres o de 32 caracteres, dependiendo del tamaño. Por lo tanto, si como variable se le asigna un literal menor, se rellenarán automáticamente los caracteres que falten mediante la función CHR$(0).

Ejemplo:

```
10  SCREEN 1
20  FOR I = 1 TO 8
30  READ B$
40  A$ = A$ + CHR$ (VAL ("&B" + B$))
50  NEXT I
60  SPRITE $ (1) = A$
70  SPRITE $ (2) = A$ + A$
80  SPRITE $ (3) = A$ + A$ + A$
90  SPRITE $ (4) = A$ + A$ + A$ + A$
100 PUT SPRITE 1, (40, 60), 15, 1
110 PUT SPRITE 2, (80, 70), 8, 2
120 PUT SPRITE 3, (120, 80), 10, 3
130 PUT SPRITE 4, (160, 90), 1, 4
140 GOTO 140
150 DATA 00111100
160 DATA 01000010
170 DATA 01000010
180 DATA 00111100
190 DATA 01000010
200 DATA 10000001
210 DATA 10000001
220 DATA 01111110
```

En este ejemplo, se han definido 4 figuras geométricas que representan gráficamente el numeral "8" que es el que aparecerá en la pantalla. La línea 30, colocada dentro un bucle de 8 rondas, apunta los datos correspondientes a la figura en la variable B$ que luego se va componiendo en la línea 40 para formar la variable literal A$ que refleja la forma de la figura a mostrar en pantalla.



Las 8 instrucciones DATA (líneas 150 a 220) fijan la figura que aparecerá en la pantalla. La línea 40 convierte primeramente el valor apuntado como valor de la variable B$ a una constante literal binaria -prefijada por &B- de la que calcula el código correspondiente y el carácter equivalente a ese código. El valor literal obtenido se va empalmando al ya contenido en la variable A$ formando la figura final.

La línea 60 designa como número 1 a la figura obtenida. En las líneas 70, 80 y 90 se designan otras tres figuras (sprites).

Si observamos ahora minuciosamente la línea 100, podemos interpretarla en la siguiente forma. Ponle la figura 1 al 'sprite' 1 (mencionado) al final de la instrucción), y que la lleve a la posición (40, 60) de la pantalla usando como color el número 15.

Hay una manera más elegante de crear el mismo efecto. En lugar de escribir 8 instrucciones DATA, y utilizar una instrucción READ dentro de un bucle FOR NEXT, podemos sustituir todo eso por una sola instrucción, en que se empalmen 8 caracteres mediante la función CHR$. Como argumento de esa función, podemos además usar notación binaria o notación hexadecimal que es más corta. Ensaya el siguiente programa:

```
10  SCREEN 1
20  A$ = CHR$(&H3C) + CHR$(&H42) +
    CHR$(&H42) + CHR$(&H3C) + CHR$(&H42)
    + CHR$(&H81) + CHR$(&H81) + CHR$(&H7E)
30  SPRITE $ (1) = A$
40  SPRITE $ (2) = A$ + A$
50  SPRITE $ (3) = A$ + A$ + A$
60  SPRITE $ (4) = AS + A$ + A$ + A$
70  PUT SPRITE 1, (40, 60), 15, 1
80  PUT SPRITE 2, (80, 70), 8, 2
90  PUT SPRITE 3, (120, 80), 10, 3
100 PUT SPRITE 4, (160, 90), 1, 4
110 GOTO 110
```

4.2.4 Instrucciones y funciones dependientes de la máquina

Nota: Las siguientes son instrucciones y funciones internas, pensadas para programadores expertos únicamente. Si no sabes lo que significan, por favor, no las uses.

4.2.4.1 OUT

Propósito: Enviar un octeto a un portal de salida del ordenador.

Versiones: Cassette, diskette.

Formato: OUT <nuroportal>, <octeto>

Observaciones: El parámetro <nuroportal> ha de ser entero y en la gama de 0 a 255. El <octeto> que se envía y deposita en dicho portal, ha de ser entero y en la gama 0 a 255.

El comando OUT es complementario de la función INP.

Ejemplo: OUT 32, 100
Envía al portal de salida designado por 32, el octeto binario equivalente al número decimal 100.

4.2.4.2 WAIT

Propósito: Detener la ejecución del programa mientras se vigila el estado de un determinado portal de entrada de la máquina.

Versiones: Cassette, diskette.

Formato: WAIT <nuroportal>, <yliado-AND-I> [,<oleado-XOR-J>]

Observaciones: El parámetro <nuroportal> es un entero en la gama 0 a 255. Los otros dos parámetros <yliado-AND-I>, y <oleado-XOR-J> son expresiones numéricas enteras en la gama 0 a 255.

La instrucción WAIT hace que se suspenda la ejecución del programa, y se entre en un bucle que vigile el estado de un determinado portal de entrada. En ese bucle, el octeto que haya en dicho portal sufre una operación XOR con el parámetro mencionado como J en la instrucción, y el resultado intermedio obtenido sufre una operación AND con el octeto mencionado como I en la instrucción. Si el resultado final de la doble operación lógica es 0, el BASIC vuelve a examinar el portal mencionado y a efectuar de nuevo la doble operación. Solamente cuando se obtiene un resultado distinto de 0, la ejecución continúa con la siguiente instrucción del programa.

Si se omite el parámetro J, se adopta como valor el 0.

Precaución: Es posible meterse en un bucle sin fin con la instrucción WAIT. Si así ocurre, la máquina tiene que volver a ponerse en marcha apagando y encendiendo.

Ejemplo:

```
WAIT, 32, 2
```

Suspenderá la ejecución del programa hasta que en el portal 32 aparezca el octeto de valor decimal 2. (Puedes comprobar que debe tener un 1 en la segunda posición).

4.2.4.3 INP

Propósito: Función cuyo resultado es el octeto depositado en un determinado portal de entrada de la máquina.

Versiones: Cassette, diskette.

Formato: INP (<nuroportal>)

Observaciones: El argumento <nuroportal> es un entero en la gama 0 a 255, y el resultado viene dado como un número decimal entero en la gama 0 a 255.

La función INP es complementaria del comando OUT.

Ejemplo: X = INP (250)

Esta instrucción examina el octeto que haya en ese momento en el portal designado como 250, y lo asigna como valor de la variable X.

APENDICE

A. RESUMEN DE MENSAJES DE ERROR Y CODIGOS DE ERROR

Si el BASIC detecta un error, suspende la ejecución del programa y presenta en pantalla un mensaje explicativo del error. Es posible detectar (atrapar con cepos) los errores y examinarlos, antes de que actúe el BASIC, incluyendo en el programa la instrucción ON ERROR y usando las variables del sistema ERR/ERL.

Todos los mensajes que BASIC muestra, tienen asociados unívocamente un código alfabético para el error y un código numérico, tal y como se listan a continuación:

Código	Número	Mensaje
NF	1	NEXT without FOR Una variable en una instrucción NEXT no corresponde con ninguna de las variables mencionadas previamente en una instrucción FOR.
SN	2	Syntax error Se ha examinado una línea de programa o un comando que contiene alguna palabra reservada mal deletreada, o paréntesis que no concuerdan, o signos de puntuación incorrectos, etc. El BASIC de Microsoft, automáticamente pasa al modo de edición sobre la línea que provocó el error.
RG	3	RETURN without GOSUB Se ha encontrato una instrucción RETURN sin que previamente se hubiera encontrado la correspondiente instrucción GOSUB.
OD	4	Out of DATA Se ha ejecutado una instrucción READ cuando ya no quedan datos en instrucciones DATA que no hayan sido apuntados como valores de variables en el programa.



FC   5   Illegal function call

Se pasa como argumento de una función numérica o litérica, un argumento que está fuera de la gama permitida. También puede producirse un error FC como resultado de:

1. Un subíndice de una variable colectiva o tabla, negativo o excesivamente grande.
2. Un argumento negativo o cero con la función LOG.
3. Un argumento negativo para SQR.
4. Una mantisa negativa con un exponente no entero.
5. Se cita una función USR, sin que previamente se le haya dado la dirección donde comienza.
6. OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO, no adecuados.

OV   6   Overflow

El resultado de un cálculo rebasa la capacidad prescrita en MBASIC para los números. Si por el contrario, ocurriera como resultado un infravalor, se tomaría 0 y la ejecución continuaría sin mensajes de error.

OM   7   Out of memory

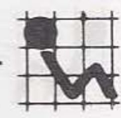
El programa es demasiado grande, o hay demasiados ficheros, o demasiados bucles, o desvíos a subrutinas, o demasiadas variables, o expresiones que son demasiado complicadas.

UL   8   Undefined line number

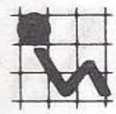
Se menciona en una instrucción GOTO, GOSUB, IF...THEN...ELSE, o DELETE, un número de línea que no está definido en el programa.

BS   9   Subscript out of range

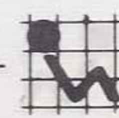
Se está haciendo referencia a un elemento de una ringla o tabla con un valor para el subíndice que cae fuera de las dimensiones, o con un incorrecto número de subíndices.



DD	10	Redimensioned array Se vuelve a dar una instrucción DIM para una tabla explícitamente dada como tal en el programa, o que el BASIC la haya generado con dimensión 10 como consecuencia de haber mencionado alguno de sus elementos.
/0	11	Division by zero Se ha encontrado en una expresión que el divisor vale 0, o en la operación de exponenciación resulta que se quiere elevar 0 a una potencia negativa. No es necesario arreglar este error, porque el programa continúa ejecutándose. El valor 'infinito' en la máquina con el signo correspondiente al número que se va a dividir, se toma como resultado de la división; o ese valor pero positivo como resultado si es la exponenciación.
ID	12	Illegal direct Se ha dado un comando que es ilegal en el modo directo, y sólo puede usarse como instrucción.
TM	13	Type mismatch Discordancia en la clase de las variables y las constantes que se le asignan, por ejemplo, una variable literal por su nombre recibe un dato numérico o viceversa; o una función espera un argumento numérico y se le entrega un argumento literal o viceversa.
OS	14	Out of string space El espacio reservado para las variables literales ha hecho que el BASIC al tratar el programa sobrepase la memoria libre disponible. El BASIC reparte el espacio para los literales en una forma dinámica, hasta que se queda sin memoria disponible.
LS	15	String too long Se intenta crear una cadena de caracteres que tiene más de 255 caracteres adosados.
ST	16	String formula too complex Una expresión literal es demasiado larga o demasiado compleja. Debiera desglosarse en expresiones más pequeñas o simples.



CN	17	Can't continue Se intenta que siga un programa después de: 1. Se detuvo la ejecución debido a una detección de error. 2. Se ha modificado durante la pausa de la ejecución. 3. No existe instrucción.
UF	18	Undefined user function Se cita una función mediante el nombre FN... antes de habersela definido al BASIC mediante la instrucción DEF FN.
	19	Device I/O error Se ha producido un error de entrada/salida al intercambiar datos con el cassette, la ductora de disco, la impresora, o la pantalla. Es un error irrecuperable ('fatal').
	20	Verify error El programa existente en memoria es diferente del programa que se está examinando en el cassette.
	21	No RESUME Se ha desviado la ejecución hacia una rutina para el tratamiento de los errores atrapados, y no contiene la instrucción final RESUME para poder reanudar la ejecución.
	22	RESUME without error Se encuentra una instrucción RESUME sin haberse producido el error que le hace desviarse a las rutinas de tratamiento.
	23	Unprintable error No existe ningún mensaje asignado a ese error producido, por lo tanto no es 'ostensible' en pantalla. Se produce habitualmente por un error que no tiene asociado un código de error.
	24	Missing operand Una expresión contiene un signo de operación sin que le siga ningún operando.



25 Line buffer overflow

Se está tecleando una línea que tiene demasiados caracteres como para caber en el 'buffer' de entrada de datos por teclado.

26 Unprintable errors

.
.
49

Estos códigos no tienen asociados mensajes y están reservados para futuras ampliaciones del BASIC.

50 FIELD overflow

Se intenta alojar más octetos en una instrucción FIELD de los que fueron especificados para la longitud del registro en un fichero 'discernial' (random). O al tratar dicho fichero como secuencial, se ha llenado el recinto de memoria reservado para FIELD.

51 Internal error

Se ha detectado un disfuncionamiento interno.

Infórmese a Microsoft -por favor- de las condiciones que produjeron la aparición del mensaje.

52 Bad file number

En una instrucción se hace mención de un fichero con un número que no ha sido reflejado en una instrucción OPEN, previa, o se sale fuera de la gama de números de ficheros limitada por la instrucción MAXFILES.

53 File not found

Se menciona un nombre de fichero en una instrucción LOAD, KILL o OPEN que no existe en el diskette.

54 File already open

Se intenta abrir un fichero que ya estaba abierto anteriormente en ese programa; o se intenta eliminar (KILL) un fichero que está abierto todavía.



55	Input past end
	Se está ejecutando una instrucción INPUT después de que todos los datos del fichero han sido ingresados en el programa, o el fichero está vacío (nulo). Para evitarlo, si es tratamiento secuencial, usa la función EOF.
56	Bad file name
	Se usa un nombre 'malo' para el fichero con instrucciones como LOAD, SAVE, KILL, NAME, etc.
57	Direct statement in file
	Se ha encontrado un comando directo (sin número de línea que lo haga instrucción) al cargar un fichero ASCII. La carga se termina.
58	File already exists
	Se intenta cambiar el nombre de un fichero en diskette mediante la instrucción NAME y se le da como nombre de fichero uno que ya está en ese diskette.
59	Sequential I/O only
	Se da una instrucción que corresponde al acceso discernial (random) y el fichero se abrió como secuencial.
60	File not OPEN
	Se intenta meter o sacar un registro en un fichero, mediante PRINT#, o INPUT#, etc. y no ha sido previamente abierto mediante la instrucción OPEN.
61 . . . 255	Unprintable error Estos códigos de error no tienen asociado ningún mensaje todavía. Están reservados para ser usados por los usuarios que definen sus propios códigos de error y los mensajes correspondientes.



DIFERENCIAS ENTRE

SV1		y	MSX	
SCREEN 0	TEXTO		SCREEN 0	TEXTO
SCREEN 1	HI-RES		SCREEN 1	TEXTO + SPRITES
SCREEN 2	MUTICOLOR		SCREEN 2	HI-RES
			SCREEN 3	MULTICOLOR

SV1:

WIDTH = anchura en caracteres
puede ser 39,40 ú 80

LOCATE = localiza el cursor
tanto en HI-RES o texto

MSX:

WIDTH = anchura en caracteres
1 → 40

LOCATE = localiza el cursor
solamente en modo Texto

VDP (N)
Retorna el valor del registro n
del VDP
N = 0....7

BASE (N)
Retorna la dirección de cada tabla
en la VRAM

n = 0 base del modo texto
n = 1
n = 2 base del generador de caracteres
n = 3
n = 4
n = 5 base del modo texto
n = 6 base del color
n = 7 base del generador de caracteres
n = 8 base de los atributos de los sprites 6912
n = 9 base de los dibujos de los sprites 14.336

SCREEN MODO, SPRITE, BAUDIOS

SCREEN 0,0,1	Graba en cassette a 1200 baudios (normal)
SCREEN 0,0,2	Graba en cassette a 2400 baudios (rápido)







EDITADO POR: indescomp

Este Libro ha sido escaneado con el único fin de **PRESERVARLO** y que pueda ser de Utilidad a cualquier Usuario de MSX pasado, presente y/o futuro.

Si este Libro te ha resultado útil piensa que si tú tienes Material que crées que pueda ser interesante y no esté preservado (Libros, Revistas, Juegos, etc) la Comunidad MSX-era te agradecería que lo PRESERVARAS y lo pusieras a disposición de todos.

Libro escaneado por José Manuel Soto y terminado de escanear el **8 de Julio de 2014**.

Para cualquier contacto:

josemanuel74@gmail.com